Video IBPster

Scott Atchley Stephen Soltesz James S. Plank Micah Beck

Logistical Computing and Internetworking Lab
Department of Computer Science, University of Tennessee
Knoxville, TN 37996

Technical Report UT-CS-02-490
Department of Computer Science
University of Tennessee
October 31, 2002.

http://www.cs.utk.edu/~plank/plank/papers/CS-02-490.html

Abstract

At iGrid 2002, members of the Logistical Computing and Internetworking Lab (LoCI) had two goals. The first was to present an application, Video IBPster, built using the tools of the Network Storage Stack that delivers DVD-quality video without dropping frames, without losing data and without specialized multi-media streaming servers. The Video IBPster demo easily played MPEG-2 video files encoded at bit-rates up to 15 Megabits/second (Mbs). The second goal was to determine performance limits when using multiple, untuned TCP streams to retrieve a striped and replicated file across a long network. Since tools built using the Network Storage Stack allow striped downloads from multiple servers in parallel and since the client machines were all connected to Gigabit Ethernet (GigE), we hoped that we would observe a linear scale up of throughput when downloading from multiple servers. Although we did see increased throughput, it was not linear.

1 Introduction

In September 2002, members from the Logistical Computing and Internetworking Lab (LoCI) at the University of Tennessee participated in iGrid2002 in Amsterdam. We had two goals: the first was to demonstrate Video IBPster, which is a multi-media streaming application built using components from the Network Storage Stack. The second goal was to push the limits of parallel, untuned TCP streams. Since our tools use servers around the globe, most of which are not under our administrative control, we cannot rely on custom TCP tuning techniques that typically require root access or modifying the kernel. Therefore, we use many parallel untuned TCP streams to achieve high performance.

The paper is organized as follows: In section 2 we provide a brief overview of the Network Storage Stack and Logistical Runtime System, which is our infrastructure testbed. This testbed has several features that make it an interesting experimental platform. In section 3 we describe in detail the downloading tool used in the iGrid demo and used to test untuned TCP parallel downloads. We review the components of the IBPster demo in section 4. In section 5, we push the use of parallel streams in pursuit of higher download throughput. We conclude in section 6.

2 The Network Storage Stack

The tools that we used for the demo and for testing are based on the Network Storage Stack, developed at the University of Tennessee [ASP+02]. The goal of the Network Storage Stack (Fig-

ure 1) is to layer abstractions of network storage that allow *writable* storage resources to be part of the wide-area network in an efficient, flexible, sharable and scalable way. Its model, which achieves all these goals for data transmission, is the IP stack, and its guiding principle has been to follow the tenets laid out by End-to-End arguments [SRC84, RSC98, BMP02]. Two fundamental principles of this layering are that each layer should (a) *abstract* the layers beneath it in a meaningful way, but (b) *expose* an appropriate amount of its own resources so that higher layers may abstract them meaningfully (see [BMP01, BMP02] for more detail on this approach).

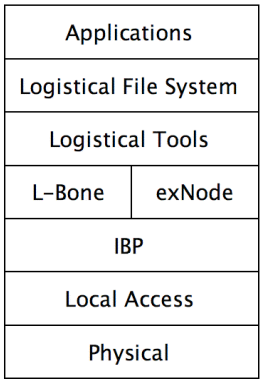

Figure 1: The Network Storage Stack

2.1 IBP

The lowest layer of the storage stack that is globally accessible from the network is the *Internet Backplane Protocol* (IBP) [PBB+01]. IBP is server daemon software and a client library that allows storage owners to insert their storage into the network, and to allow generic clients to allocate and use this storage. The unit of storage is a time-limited, append-only byte-array. With IBP, byte-array allocation is like a network **malloc()** call — clients may request an allocation from a specific IBP storage server (or *depot*), and if successful, are returned trios of cryptographically secure text strings (called "capabilities") for reading, writing and management. Capabilities may be used by *any* client in the network, and may be passed freely from client to client, much like a URL.

IBP does its job as a low-level layer in the storage stack. It abstracts away many details of the underlying physical storage layers: block sizes, storage media, control software, etc. However, it also exposes many details of the underlying storage, such as network location, network transience and the ability to fail, so that higher layers in the stack may abstract these more effectively.

2.2 L-Bone and exNode

While individual IBP allocations may be employed directly by applications for some benefit [PBB+01], they, like IP datagrams, benefit from some higher-layer abstractions. The next layer contains the *L-Bone*, for resource discovery and proximity resolution, and the *exNode*, a data structure for aggregation. Each is defined here.

The L-Bone (Logistical Backbone) is a distributed runtime layer that allows clients to perform IBP depot discovery. IBP depots register themselves with the L-Bone, and clients may

then query the L-Bone for depots that have various characteristics, including minimum storage capacity and duration requirements, and basic proximity requirements. For example, clients may request an ordered list of depots that are close to a specified city, airport, US zipcode, or network host. Once the client has a list of IBP depots, it may then request that the L-Bone use the Network Weather Service (NWS) [WSH99] to order those depots according to bandwidth predictions using live networking data. Thus, while IBP gives clients access to remote storage resources, it has no features to aid the client in figuring out which storage resources to employ. The L-Bone's job is to provide clients with those features.

The exNode is a data structure for aggregation, analogous to the Unix inode (Figure 2). Whereas the inode aggregates disk blocks on a single disk volume to compose a file, the exNode aggregates IBP byte-arrays to compose a logical entity that may be used like a file. Two major differences between exNodes and inodes are that the IBP buffers may be of any size, and their extents may overlap and be replicated. Thus, the exNode allows users and applications to create network files out of time-limited and failure-prone IBP allocations in such a way that much stronger properties (e.g. fault-tolerance, longer durations) may be achieved. ExNodes are represented by XML encodings, manipulated by an exNode library. Like IBP capabilities, they may be passed from client to client, anywhere in the network, with no registration from a central authority.

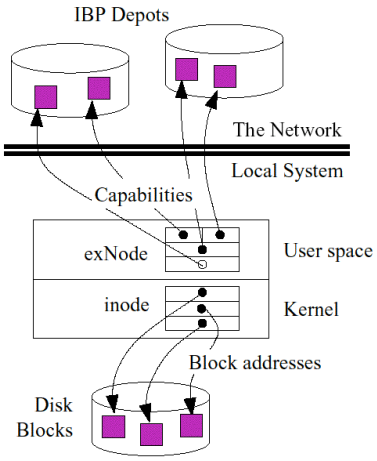

Figure 2: The exNode in comparison to the Unix inode

2.3 Logistical Runtime System

The next level in the stack contains tools and a client library that compose the Logistical Runtime System (LoRS). These tools allow users to create, manipulate and use the network "files" supported by the exNode. These files reside on IBP depots located by the L-Bone. The functionalities supported by LoRS are:

Upload: Create a network file from a local file, input stream or memory buffer.
Download: Get the bytes from a network file and store them locally or stream them to an application.
Augment: Add more replicas to a network file.
Trim: Subtract replicas from a network file.

Refresh: Extend the time limits of the IBP allocations.

Note that both **upload** and **augment** allow the user to stripe and replicate the file in a very flexible manner. Moreover, **augment** and **trim** allow the user to *route* the file from one network location to another.

3 Striped Downloads from Multiple Servers

Because the network is inherently unreliable, the exNode allows multiple replicas to improve fault-tolerance [ASP+02]. The exNode library places no restrictions on the number of replicas, nor do replicas need to be complete copies of the data. There are many algorithms available when retrieving replicated data. The simplest choice is selecting a single replica and downloading it completely from one source (e.g. [RWE+01, CHM+02]). The downside to this approach is determining which replica to use.

Another option is an adaptive algorithm [PAD+02]. In this case, the data is divided into small blocks. Next, *t* threads are assigned to each of the sources. The threads alternate while selecting the next available block from the queue. When a thread finishes it, it selects the next block in the queue. If a download fails, a thread from another source will try to download the block. This provides improved fault-tolerance. Also, work is under way to add forward error-correction using erasure codings to exNodes for additional fault-tolerance [Pla97, WK02, Riz97].

This algorithm is adaptive, because IBP servers with high bandwidth to the client should download many more blocks than those with low bandwidth. Moreover, as long as there are many blocks to be downloaded, the algorithm may adapt to fluctuating network conditions. The selection of the block size is of concern. Blocks that are too small may suffer too much from the effects of latency and overhead during their downloads, while blocks that are too large may hinder the degree of adaptive load-balancing that the algorithm may achieve. This was the algorithm used at iGrid for the demo and for the tests in section 5.

A more recent algorithm developed by the LoCI Lab is the *progress-driven redundancy* [PAD+02], which performs much better than the adaptive algorithm. Because there was such an improvement in performance, we have modified the download tools to use this algorithm.

The LoRS download tool provides a high degree of flexibility for the user. The user can control the number of threads used, the transfer block size, how much memory is used to buffer downloaded blocks that are not ready to be released, and how much data to pre-buffer before releasing to a streaming application. The tools also allow downloading of a sub-extent of the data.

The number of threads determines how many simultaneous TCP connections the download tool can make. If the user specifies too few connections, the throughput will suffer. If the user specifies too many connections, the tool may experience too much context-switch overhead, the IBP depots holding the data may be overwhelmed by connection requests, the link can become congested or zealous sysadmins may think that their network is being used in a denial of service (DOS) attack. Also, using many TCP streams is considered unfair, because during congestion or packet loss, only a single stream may back off instead of the entire flow. We have previously discussed the difficulty of balancing higher performance through multiple TCP streams with good network citizenship [PAD+02].

As mentioned above, choosing the “right” block size can dramatically affect the performance of the download. To take advantage of the basic download algorithm, we typically use a block size of 512 KB, although we have seen good results with blocks up to 2 MB.

When downloading, each thread has a buffer equal in size to a single block. Since the threads are not guaranteed to finish their transfer in order, the thread must hold its data until all threads downloading previous data finish. Then it can release its block. Since this will lead to idle threads and lost work, the user can specify the size of a waiting queue that will hold completed blocks and free the thread to begin downloading a new block. Once all the previous blocks are downloaded, a block in the waiting queue will be released. Ideally, the waiting queue will hold the entire data file. In reality, the user will specify as much memory as possible that is less than the physical size of the system's memory. If the user specifies a queue that is larger than the physical memory, then performance will suffer dramatically due to thrashing.

Typically, once blocks are downloaded, the tool releases them immediately. When streaming to a multi-media player, however, the user may want the download tool to hold a number of the first blocks and release them all together. The download tool allows the user to specify how many blocks to pre-buffer. This does not use additional memory; it merely has the output thread wait before releasing any data.

4 Video IBPster

The Video IBPster demo combines the LoRS command line tools and open-source audio and video players. For the demo, the user runs two TCL/TK graphic user interfaces (GUI). One provides control of the command line LoRS tools and the other is a map, which represents the L-Bone with locations of IBP depots (Figure 3).

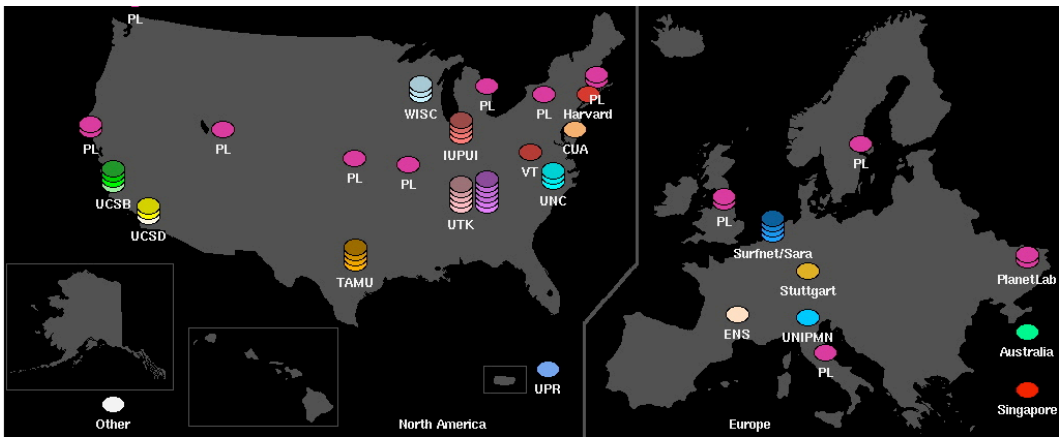

Figure 3: L-Bone map with IBP depots

First, the user uploads a file and creates an exNode (Figure 4). In this example, the file is being stored to IBP depots at UTK. This copy has 10 pieces stored over 8 machines. All pieces are uploaded in parallel. The outlined blocks represent IBP allocations that are receiving data. When the block transfer is complete, the block is filled in with the color of the IBP depot that receives it.

After the upload completes, the user typically adds more replicas to the exNode. In this example, a replica has already been added at UNC and another is being added at UCSB (Figure 5). Note that all ten pieces are being copied simultaneously. Currently, the tool does not try to determine which replica would provide the best performance as the source of the copy. That is part of our ongoing research.

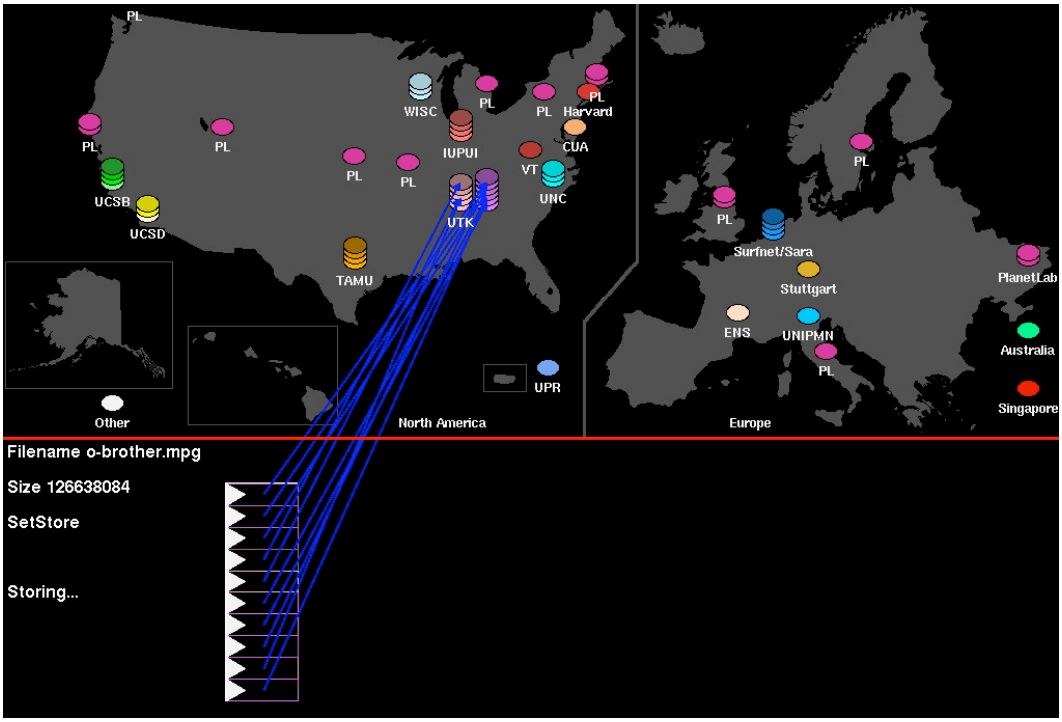

Figure 4: Upload in progress

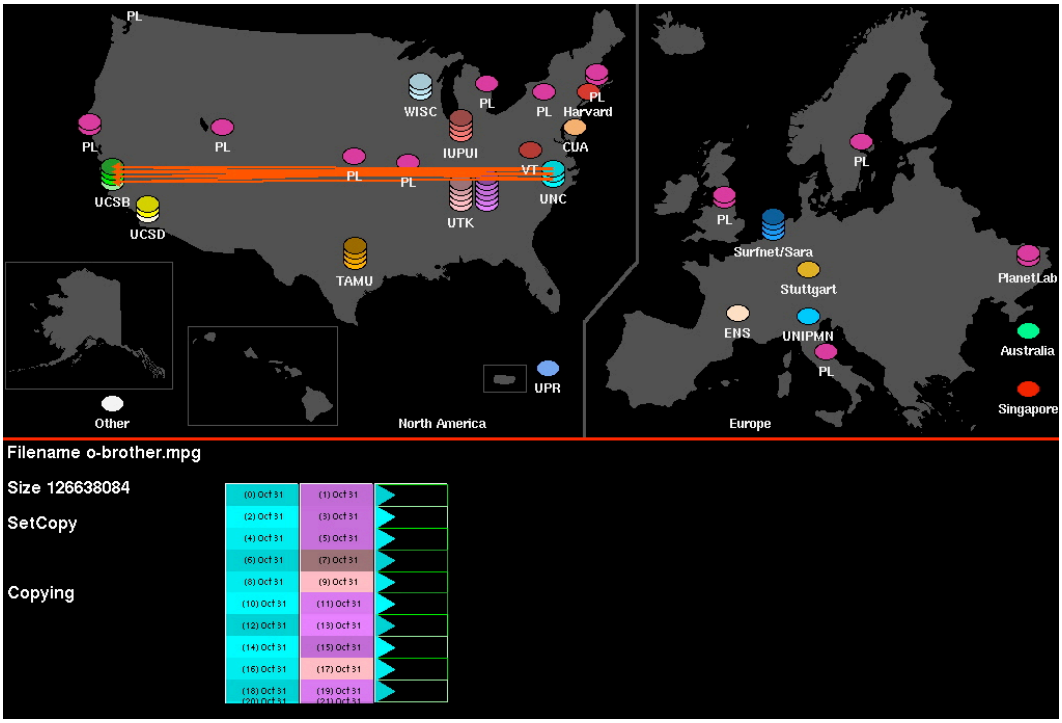

Figure 5: Augment adding replica in UCSB

Lastly, the user can download the file to disk or to a multi-media player capable of reading data on stdin (e.g. mplayer, mpg123, VideoLan Client, etc). In this case, the stored file is 126 MB (Figure 6). The download tool is downloading eight 2 MB blocks in parallel from multiple depots at UTK. As a block is being downloaded, it is represented by a hollow rectangle in the lower right. When the data is downloaded, the block is filled in. The white column on the lower, far right represents data blocks that have been downloaded and released to the disk or player.

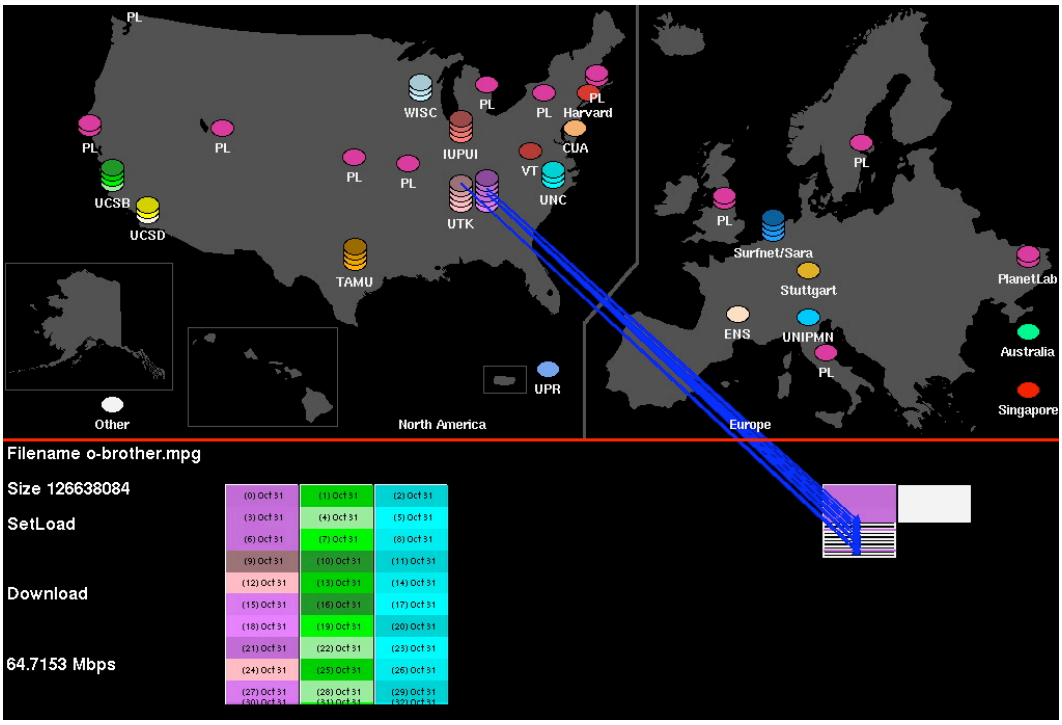

Figure 6: Downloading 2 MB blocks in parallel

During iGrid, we successfully streamed 15 Megabit/second (Mbps) video from the US. Since the LoRS tools are designed to move data files, not just multi-media, they use TCP for transfers. Because of this, IBPster does not drop frames. If the download tool does not supply data to the player fast enough, the audio or video will pause until it receives more data. If this happens several times in a row, the player will appear to stutter. We found that 10-15 threads downloading 512 KB blocks from 8-10 IBP depots would stream data to the player fast enough for 15 Mbps video.

5 Testing the Limits of Trans-Atlantic Untuned TCP

In order to achieve high throughput using TCP, users have three options: tuning the TCP stack [DMT02, TGL+01], modifying the kernel [FF01, SMM98], or using parallel streams [KT01, SBG00, Fer02, HAN02, TJ+94]. The first two options require administrative control of the machine, while the third does not.

All of the software that composes the Network Storage Stack is available online (http://loci.cs.utk.edu) and participation is open to everyone. Since, we do not have administrative control over many of the machines registered with the L-Bone, we cannot rely on either custom TCP tuning techniques or modifying the kernel. To achieve high performance, we can

either use many parallel, untuned TCP streams or we can use some form of UDP transfer. Currently, the LoRS tools use TCP by default and we are working on incorporating UDP transfer as an option when augmenting. While we had access to the iGrid network, we wanted to test the limits of throughput using parallel, untuned TCP streams.

To test performance, we stored a 276 MB file into three exNodes. The first exNode had three replicas in the US (IN, NC and MO), the second had three replicas in Europe (UK, IT, SE) and the third had a single replica on one of two Linux clusters at Sara (University of Amsterdam), which is on the same LAN as the client. All the IBP depots had 64 threads available for data transfers. The client machine was the other Sara Linux 4-way cluster.

In the US, we have consistently achieved 70-80 Mbps between one server and one client, both of which have 100 Mbps network interface cards (NICs). At iGrid with the GigE LAN, the trans-Atlantic 25 GB/s network and a GigE equipped client, we hoped that we would be able to see some linear scaling of throughput when drawing from multiple IBP depots. Although we did see higher throughput, it was nowhere near linear. In fact, it peaked at 103 Mbps when downloading from the first exNode that had three depots in the US (Figure 7). Although the three IBP depots had a combined total of 196 threads available, the client reached the peak performance with 150 threads. The US depots averaged 188 ms round-trip time from the client when pinged.

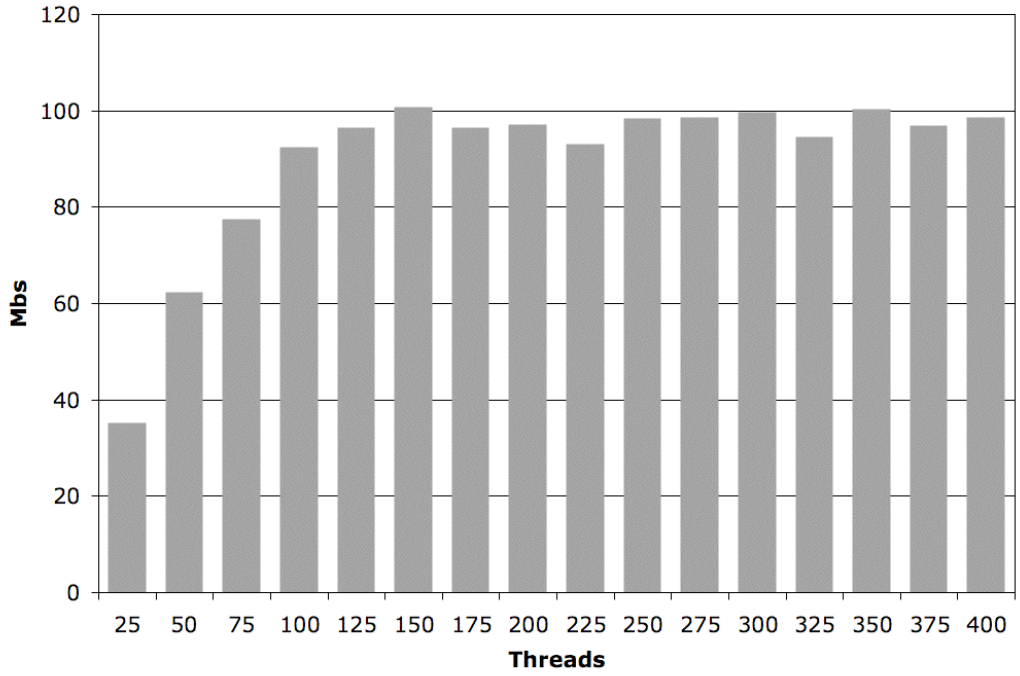

Figure 7: Throughput when downloading from US

For comparison, we repeated the test using the second exNode with three European IBP depots. We tested downloads from a single IBP depot (UK) to establish the throughput for a one-to-one transfer. The peak throughput average 89 Mbps when using 25 threads. Next, we downloaded from all three depots. This time the throughput was higher, but still well short of a linear speedup (Figure 8). Using 275 threads, throughput averaged 184 Mbps. The European clients round-trip times averaged between 14 and 45 ms when pinged.

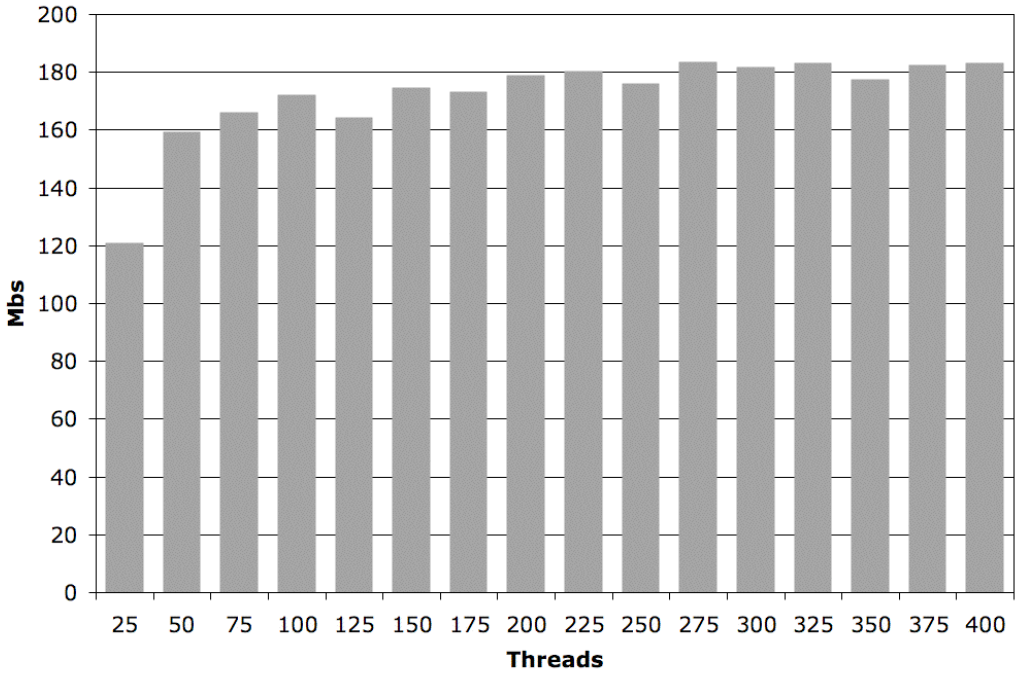

Figure 8: Throughput when downloading from Europe

Concerned that the LoRS tools may be the limiting factor, we downloaded across the Sara LAN. In this test, the IBP depot was running 96 threads, but it only needed 15 to reach its peak performance of 749 Mbps (Figure 9). The tools did not seem to be the limiting factor.

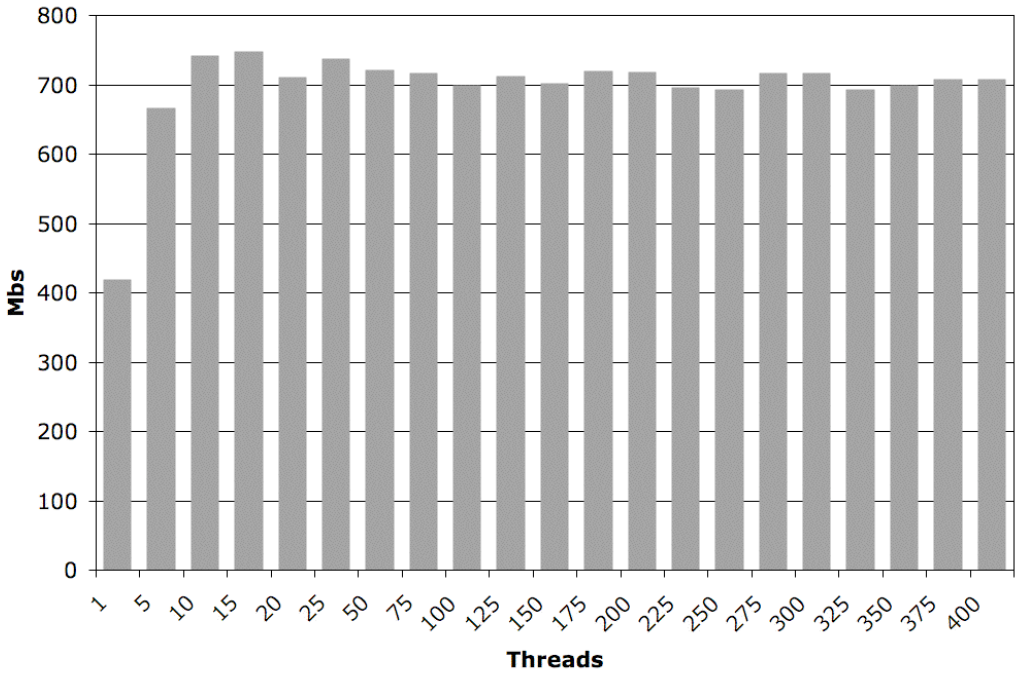

Figure 9: Throughput when downloading across Sara LAN

6 Conclusion

By participating in the iGrid conference, the LoCI Lab was able to demo an application easily built using components of the Network Storage Stack. Video IBPster was able to stream DVD-quality video using parallel, untuned TCP streams. This demo does not require proprietary or specialized servers. Instead, it shows the flexibility and the power of the design of the Network Storage Stack.

Participation at iGrid also allowed us to test the limits of untuned TCP. In many applications, the users will not have control of the machines in order to perform TCP tuning or kernel modifications. In these cases, they will resort to using untuned TCP for reliable transfer or abandon TCP altogether. While acknowledging the concerns regarding using multiple TCP streams, this approach can produce improved performance. When moving to GigE capable networks and machines, using multiple streams will improve performance, but it will not scale linearly.

7 Acknowledgements

This material is based upon work supported by the National Science Foundation under grants ANI-0222945, ANI-9980203, EIA-9972889, EIA-9975015 and EIA-0204007, the Department of Energy under grant DE-FC02-01ER25465, and the University of Tennessee Center for Information Technology Research. The authors greatly acknowledge Alex Bassi and Xiang Li who provided their time and intimate knowledge of the IBP, Jeremy Millar who provided the same with the exNode library, Yong Zheng for his insights into the LoRS tools, and the organizers of iGrid2002. The authors also acknowledge Paul Wielinga at Sara for access to their Itanium clusters, Rich Wolski, Graziano Obertelli, Norman Ramsey, Hunter Hagewood and Planet Lab for depot access. The authors also gratefully thank Terry Moore for all his contributions to the LoCI.

References

[ASP+02] S. Atchley, S. Soltesz, J. S. Plank, et al. Fault-Tolerance in the Network Storage Stack, *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems* in conjunction with *International Parallel and Distributed Processing Symposium*, Ft. Lauderdale, FL, April, 2002. http://www.ipdps.org/.

[BMP01] M. Beck, T. Moore, and J. S. Plank. Exposed vs. encapsulated approaches to grid service architecture. In *2nd International Workshop on Grid Computing*, Denver, November 2001. http://www.gridcomputing.org/grid2001.

[BMP02] M. Beck, T. Moore, and J. S. Plank. An end-to-end approach to globally scalable network storage. In *ACM SIGCOMM '02*, Pittsburgh, August 2002.

[CHM+02] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[DMT02] T. Dunigan, M. Mathis, and B. Tierney. A TCP tuning daemon. In *SC02: High Performance Networking and Computing Conference*, Baltimore, 2002.

[Fer02] G. Ferrache. bbftp: On-line documentation for release 2.2.1. http://doc.in2p3.fr/bbftp/doc.2.2.1.html, Documentation from the IN2P3 Computing Center, Lyon, France, August 2002.

[FF01] M. Fisk and W. Feng. Dynamic Right-Sizing in TCP. In *Los Alamos Computer Science Institute Symposium*, 2001.

[HAN02] T. Hacker, B. Athey, and B. Noble. The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network. *Proc. 16th IEEE-CS/ACM International Parallel and Distributed Processing Symposium (IPDPS)*, 2002. URL: http://www.cnaf.infn.it/ ferrari/papers/tcp/IPDPS.pdf.

[KT01] Rajesh Kalmady and Brian Tierney. A Comparison of GSIFTP and RFIO on a WAN , March 2001. http://grid-data-management.web.cern.ch/grid-data-management/publications.html.

[PAD+02] J. S. Plank, S. Atchley, Y. Ding and Micah Beck, Algorithms for High Performance, Wide-Area, Distributed File Downloads, *Technical Report CS-02-485*, University of Tennessee Department of Computer Science, October 8, 2002.

[PBB+01] J. S. Plank, A. Bassi, M. Beck, T. Moore, D. M. Swany, and R. Wolski. Managing data storage in the network. *IEEE Internet Computing*, 5(5):50–58, September/October 2001.

[Pla97] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[Riz97] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM SIGCOMM Computer Communication Review*, 27(2):24-36, 1997.

[RSC98] D. P. Reed, J. H. Saltzer, and D. D. Clark. Comment on active networking and end-to-end arguments. *IEEE Network*, 12(3):69–71, 1998.

[RWE+01] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5):40–49, 2001.

[SBG00] H. Sivakumar, S. Bailey and Robert L. Grossman. PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *Supercomputing SC2000*, 2000. http://www.sc2000.org/techpapr/papers/pap.pap240.pdf.

[SMM98] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. In *SIGCOMM*, pages 315–323, 1998.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems,*, 2(4):277–288, November 1984.

[TGL+01] B. L. Tierney, D. Gunter, J. Lee, M. Stoufer. Enabling Network-Aware Applications, *Proceedings of IEEE HPDC '01*, San Francisco, California, August 7-9, 2001. http://www-itg.lbl.gov/HPDC-10.

[TJ+94] B. Tierney,W. Johnston, and et. al. Distributed parallel data storage systems: A scalable approach to high speed image servers. In *Proceedings of the ACM Multimedia Symposium*, June 1994.

[WK02] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, March 2002.

[WSH99] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, 1999.