Improving Performance in the Network Storage Stack

Scott Atchley James S. Plank Zheng Yong Ding Jin Long Zhou
Stephen Soltesz Micah Beck Terry Moore[*]

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

http://loci.cs.utk.edu
[atchley,plank,yong,djin,zlong,soltesz,mbeck,tmoore]@cs.utk.edu

Technical Report UT-CS-02-481
Department of Computer Science
University of Tennessee
April, 2002

Abstract

This paper addresses the issue of improving performance when using multi-threading in network storage applications. An abstraction of network storage called the *Network Storage Stack* is detailed along with the software layers (IBP, the L-Bone, exNode, and Logistical Tools) that have been developed to implement it. These layers have been implemented so that applications use single connections to access and utilize network storage. In this paper, we explore the benefits of adding multi-threading to to the applications at various points. We perform experiments utilizing network storage both on local-area clusters and on the wide-area. As expected, multi-threading improves performance, but also leads to other challenges in implementation and performance tuning.

1 Introduction

Whether computing on local clusters or on the wide-area grid, the ability to move and store data is critical. Grid computing usually requires moving and storing large amounts of data from a few gigabytes to hundreds of terabytes. For example, a researcher using the grid may need to pre-position data near the computation site or his application may need to write temporary checkpoints during the computation to improve fault-tolerance. Regardless of the distributed computing environment, researchers have two primary requirements for handling the logistics of their data:

- When moving data, do so as quickly as possible, and
- Wherever the data is stored, ensure the ability of data access.

The **Lo**gistical **C**omputing and **I**nternetworking (**LoCI**) Lab at the University of Tennessee has been working to change the view of storage in the network to improve its functionality, performance, scalability and reliability. As such, the LoCI Lab has been demonstrating the power of *Logistical Networking* in cluster and wide-area settings.

Logistical Networking takes the rather unconventional view that storage can be used to augment data transmission as part of the unified network resource framework, rather than being used simply as a network-attached resource. We use the term "logistical" to draw an analogy to transportation and military logistics. In the these environments, logistics combine the use of long-haul transportation and warehousing for moving raw materials and finished goods.

[*]This material is based upon work supported by the National Science Foundation under grants ACI-9876895, EIA-9975015, EIA-9972889, ANI-9980203, the Department of Energy under the SciDAC/ASCR program (DE-FC02-01ER25465), and the University of Tennessee Center for Information Technology Research.

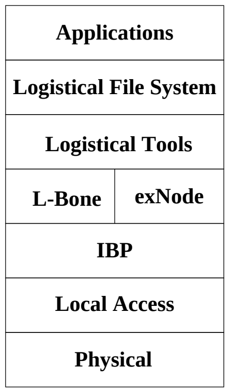

Figure 1: The Network Storage Stack

Our design for the use of network storage revolves around the concept of a *Network Storage Stack* (Figure 1). Its goal is to layer abstractions of network storage to allow storage resources to be part of the wide-area network in an efficient, flexible, sharable and scalable manner. Its model, which achieves all these goals for data transmission, is the IP stack. Our guiding principle has been to follow the tenets laid out by the End-to-End arguments [SRC84, RSC98]. 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 resources so that the higher layers may abstract them meaningfully (see [BMP01] for more detail on this approach).

In this paper, we describe the currently implemented layers of the Network Storage Stack, and how they achieve their goals. We then focus our attention on the performance of some of the components, and in particular, how that performance may be improved by multi-threading connections. Our tests involve both local-area cluster environments, and wide-area scenarios involving storage depots scattered across the United States. As expected, multi-threading improves performance, but to differing degrees depending on the environment and the aggressiveness of multi-threading. It also leads to other challenges in implementation and performance tuning.

2 The Network Storage Stack

In this section, we describe the middle three layers of the Network Storage Stack. These are the layers currently implemented by our project. All code described in this section is available via the URL `http://loci.cs.utk.edu`. The bottom (Physical) layer of the Network Storage Stack is simply the hardware (disk, RAM), and the Local Access layer is the operating system. The top two layers, while interesting, are future functionalities to be built when we have more understanding about the middle layers. In the remainder of this section, we give motivational, descriptive and implementation details about the middle three layers of the stack.

2.1 IBP

IBP stands for the *Internet Backplane Protocol*. It is lowest level of the storage stack that provides network accessibility. IBP is composed of a server daemon and a client library that allows storage owners to insert their storage into the network, and to allow generic clients to allocate and make use of 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 request an allocation from a specific IBP storage server (or *depot*), and if successful, three cryptographically secure text strings (called *capabilities*) are returned, one each 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 these may be abstracted more effectively by higher layers in the stack. More information on IBP may be found at `http://loci.cs.utk.edu/ibp/`, and in citations [PBE$^+$99, PBB$^+$01].

2.2 The L-Bone and exNode

While individual IBP allocations may be employed directly by applications for some benefit [EPBW99, 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 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, she may query the Network Weather

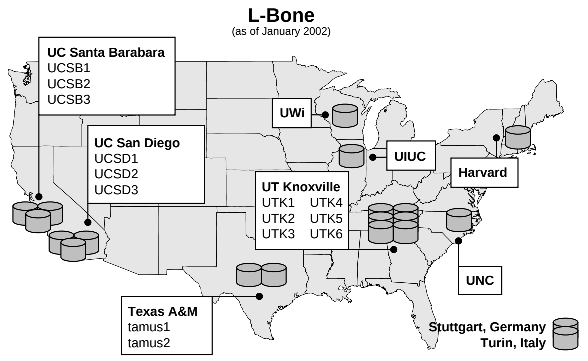

Figure 2: The L-Bone

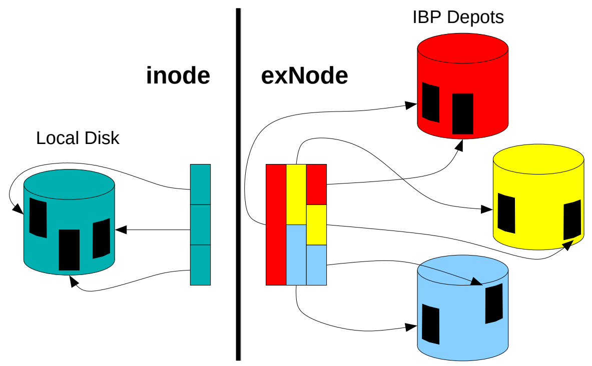

Figure 3: The exNode in comparison to the Unix inode

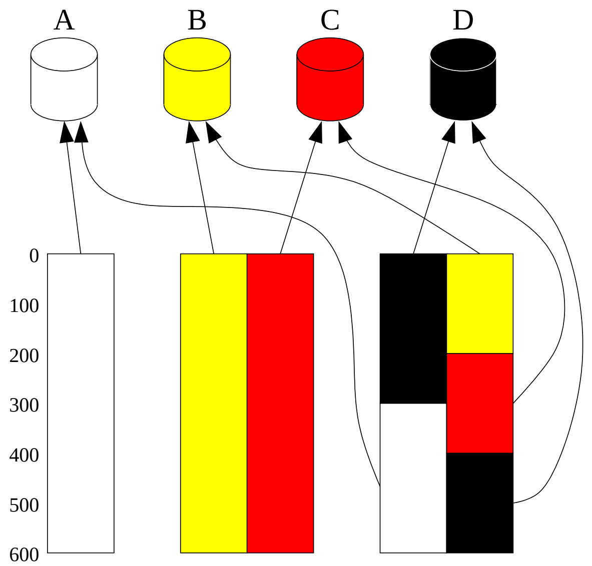

Figure 4: Sample exNodes of a 600-byte file with different replication strategies.

Service (NWS) [WSH99] to provide live performance measurements and forecasts and decide how best to use the depots.

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. As of January 2002, the L-Bone is composed of 21 depots in the United States and Europe, serving roughly a terabyte of storage to Logistical Networking applications (Figure 2). The L-Bone code and current composition may be obtained at `http://loci.cs.utk.edu/lbone/`.

The exNode is a data structure for aggregation, analogous to the Unix inode (Figure 3). 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 like a file. Two major differences between exNodes and inodes are that the IBP buffers may be of any size, and the extents may overlap and be replicated. For example, Figure 4 shows three exNodes storing a 600-byte file. The leftmost one stores all 600 bytes on IBP depot A. The center one has two replicas of the file, one each on depots B and C. The rightmost exNode also has two replicas, but the first replica is split into two segments, one on depot A and one on depot D, and the second replica is split into three segments, one each on depots B, C, and D.

In the present context, the key point about the design of the exNode is that it allows us to create storage abstractions with stronger properties, such as a network file, which can be layered over IBP-based storage in a way that is completely consistent with the exposed resource approach.

Since our intent is to use the exNode file abstraction in a number of different applications, we have chosen to express the exNode concretely as an encoding of storage resources (typically IBP capabilities) and associated metadata in XML. Like IBP capabilities, these serializations may be passed from client to client, allowing a great degree of flexibility and sharing of network storage. The use of the exNode by varying applications provides interoperability similar to being attached to the same network file system. The exNode support libraries are available at `http://loci.cs.utk.edu/exnode/`.

Logistical Tools

At the next level of the Network Storage Stack are tools that perform the actual aggregation of network storage resources, using the lower layers of the Network Stack. While the lower levels offer a variety of functionalities, this level is the first one that starts treating the composition of IBP depots as a unified logistical network resource fabric.

Basic functionalities of these tools are:

Upload: This takes local storage (e.g. a file, or memory), uploads it into the network and returns an exNode. This upload may be parameterized in a variety of ways. For example, the client may partition the storage into multiple blocks (i.e. stripe it) and these blocks may be replicated on multiple IBP servers for fault-tolerance and/or proximity reasons. Moreover, the user may specify proximity metrics for the upload, so the blocks have a certain network location.

Download: This takes an exNode as input, and downloads a specified region of the file into local storage. This involves coalescing the replicated fragments of the file, and must deal with the fact that some fragments may be closer to the client than others, and some may not be available (due to time limits, disk failures, and standard network failures).

Download is written to check and see if the Network Weather Service [WSH99] is available locally to determine the closest depots. If so, then NWS information is employed to determine the download strategy: The file is broken up into multiple extents, defined at each segment boundary. For example, the rightmost file in Figure 4 will be broken into four extents – (0,199), (200-299), (300-399), and (400-599). Then the download proceeds by retrieving each extent from the closest depot. If the retrieval times out, then the next closest depot is tried, and so on.

If the NWS is not available, then the download looks for static, albeit suboptimal metrics for determining the downloading strategy. If desired, the download may operate in a streaming fashion, so that the client only has to consume small, discrete portions of the file at a time.

Stat: Much like the Unix **stat()** system call, this takes an exNode as input and provides information about the metadata stored therein. Such information includes the stored file's name and size, and metadata about each segment or fragment of the file, including byte offset, length, IBP host ID, available bandwidth as reported by the NWS, and expiration time.

Refresh: This takes an exNode as input, and extends or reduces time limits of the IBP allocations that compose the exNode.

Augment: This takes an exNode as input, adds more replica(s) to it (or to parts of it), and returns an updated exNode. Like **upload**, these replicas may have a specified network proximity.

Trim: This takes an exNode, deletes specified fragments, and returns a new exNode. These fragments may be specified individually, or they may be specified to be those that represent expired IBP allocations. Additionally, the fragments may be only deleted from the exNode, and not from IBP.

The Logistical Tools are much more powerful as tools than raw IBP capabilities, since they allow users to aggregate network storage for various reasons:

Capacity: Extremely large files may be made from smaller IBP allocations. It fact, it is not hard to visualize files that are hundreds of gigabytes in size, split up and scattered around the network.

Striping: By breaking files into small pieces, the pieces may be downloaded simultaneously from multiple IBP depots, which may perform much better than downloading from a single source.

Replication for Caching: By storing files in multiple locations, the performance of downloading may be improved by downloading the closest copy.

Replication for Fault-Tolerance: By storing files in multiple locations, the act of downloading may succeed even if many of the copies are unavailable. Further, by breaking the file up into blocks and storing error correcting blocks calculated from the original blocks (based on parity as in RAID systems [CLG$^+$94] or on Reed-Solomon coding [Ber68, MS77, Pla97]), downloads can be robust to even more complex failure scenarios.

Routing: For the purposes of scheduling, or perhaps changing resource conditions, **augment** and **trim** may be combined to effect a routing of a file from one network location to another. First it is augmented so that it has replicas near the desired location, then it is trimmed so that the old replicas are deleted.

Therefore, the Logistical Tools enable users to store and retrieve data as replicated and striped files in the wide area, flexibly and efficiently. They may be obtained at `http://loci.cs.utk.edu/exnode/tools.html`.

3 Performance Enhancements of IBP and the Tools

The initial versions of the Logistical Tools have been written to get the functionalities working. As such, very simplistic design decisions were made. Two important examples concern multi-threading and scheduling. The first versions of the tools use one flow of control, and therefore serialize the operations, even though the existence of multiple network paths from the client to individual servers, and to multiple servers, would argue for multiple flows of control (i.e. threads) to perform operations simultaneously. Additionally, besides using the NWS to select servers for downloading decisions, no other scheduling decisions are made by the tools.

In the experiments below, we assess the benefits of multi-threading to deserialize the tools' operations, both in the local area and in the wide area. Additionally, we compare two simple strategies for scheduling uploads in the wide area. Both experiments demonstrate how design decisions of both the tools, and of IBP, impact performance. The experiments should also serve to help us understand the intricacys of programming applications on clusters and in the wide area.

Of importance in the experiments is the method by which IBP servers service simultaneous connections. IBP maintains a thread pool, composed of a fixed number of threads. The number of threads is determined by the storage owner, who may select a large number for better server performance, or a low number to limit server resource usage. Therefore, the degree to which IBP may service simultaneous connections may be restricted, and as seen below, these restrictions can impact the performance of the Logistical Tools.

Previous performance experiments of the Logistical Tools have been presented in [ASP$^+$02]. These experiments were performed using the single-threaded versions of the tools, and demonstrated the fault-tolerant properties of replicating exNodes. The numbers in that paper are consistent with the single-threaded numbers in this paper.

In the tests below, all file and download sizes are given in megabytes (MB), while all bandwidth numbers are given in megabits per second (Mb/s). All tests are the averages of at least ten runs for each data point. The tests were performed at various times throughout the day on unreserved machines and networks.

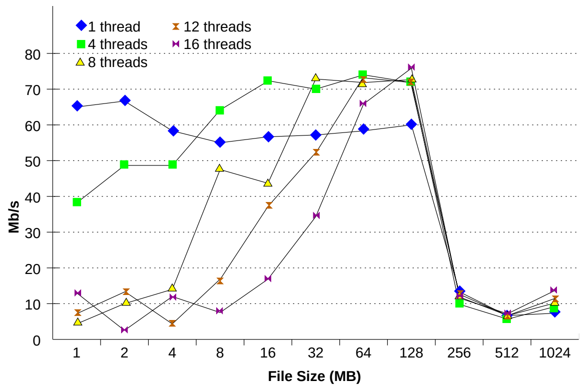

Figure 5: Multi-threaded download bandwidth in the LAN.

3.1 Test 1 - Multi-Threaded Downloads From One Server

In the first set of tests, we tested the impact of multi-threading downloads from one server to one client. The primary purpose of this test was to assess the performance of varying numbers of threads. The secondary purpose was to establish a benchmark for our multi-threaded, multi-server tests.

The tests were executed on both a local area network (LAN), and on a wide-area testbed. For the LAN test, the IBP server was a dual processor, Solaris machine with 512 MB of RAM, running 40 threads. The client was a single processor, Linux machine with 256 MB of RAM. Both machines had 100BaseT network cards.

For the first test, we download from allocations ranging from 1 MB to 1 GB using 1 to 16 threads in increments of four. The work is divided evenly between the threads and the file is stored to client memory using the **mmap()** system call. The results are displayed in Figure 5. In this test, a single thread performs best from 1 to 4 MB and while four threads show the highest bandwidth for file sizes over 4 MB. When using more threads, it takes larger files sizes to to achieve a higher bandwidth, due to the fact that multiple TCP connections need to be conditioned. The performance of all tests shows a dramatic fall-off as the size of the file reaches the size of physical memory, causing the OS to start swapping.

For the wide area version, we used an IBP depot at Texas A&M, running Linux with 980 MB of RAM and 32 threads. Here we employ up to 32 threads, and see continued improvement due to multiple network paths as more threads are utilized. We do not try more than 32 threads because of the IBP depot's thread limit. Again, at the 256 MB file size, we see a dramatic drop in per-

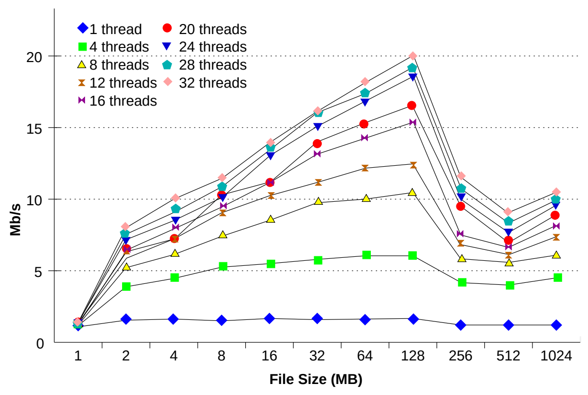

Figure 6: Multi-threaded download bandwidth in the WAN.

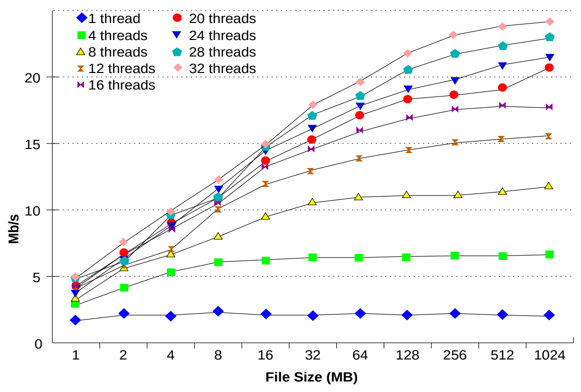

Figure 8: Multi-threaded download bandwidth in the WAN without disk I/O.

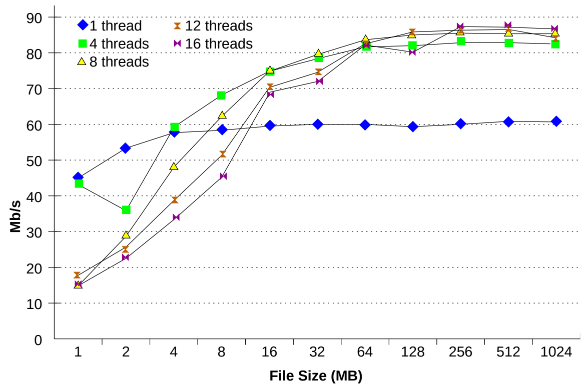

Figure 7: Multi-threaded download bandwidth in the LAN without disk I/O.

formance due to virtual memory swapping on the client machine (Figure 6).

The amount of speedup is not linearly proportional to the increase in threads. For example, while downloading the 128 MB file, although the number of threads increases from 28 to 32 threads (14%), we only observe a 6% increase in throughput. This displays diminishing returns as more threads are added.

The bandwidth numbers presented in this figure are significant in that they do improve upon the performance of standard storage transfer methods. For example, the peak measured performance of a standard FTP transfer between the same client and server was 3.5 Mb/s.

Since we observed such large drops in performance as the file size matched the amount of physical RAM, we repeated both the LAN and WAN tests and did not save the data to the disk. This time, both the LAN and WAN results show continued performance up to the 1G file size. In the LAN test (Figure 7), four threads again outperform more threads up to the 32 MB file size, and beyond that, all the multi-threaded runs were approximately equal, limited only by the performance of the machines' networking hardware.

In the WAN test (Figure 8), we see continued gains with more threads although again with diminishing returns. We notice a leveling of performance as the file size reached the size of the RAM. A question for further study is whether performance will continue to improve if more RAM is available at the client. A second question is what the upper limit of simultaneous connections between the client and server. Here the limit is set by the IBP depot. Were that limit higher (or unbounded), it is an interesting question to see at what number of threads the performance ceases to improve.

3.2 Test 2 - Multi-Threaded Uploads to One Server

Using the same testing environment as in Test 1, we tested the performance of multi-threading uploads. We omit the results for brevity, since they are very similar to the download results.

3.3 Test 3 - Multi-Threaded Downloads from Multiple Servers

For these tests, we tested the performance of downloading exNodes of files that have been partitioned into multiple fragments, where each fragment is replicated, and replicas are stored on a variety of IBP depots. These tests are therefore inherently more complex than the previous tests because scheduling decisions must be made concerning which depots should be selected for downloading, and how many threads should be allocated to

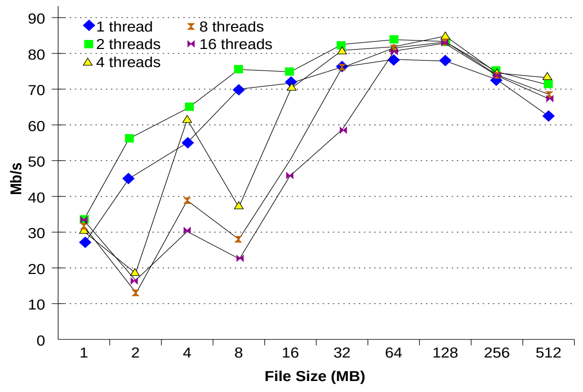

Figure 9: Multi-threaded download bandwidth from multiple servers in the LAN.

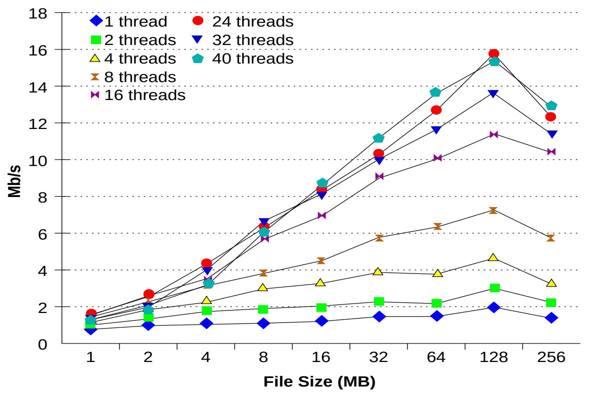

Figure 10: Multi-threaded download bandwidth from multiple servers in the WAN.

each fragment. These decisions are made with the help of the Network Weather Service, which improves the decision of server selection, but also has an inherent overhead.

In each of these tests, our exNode files are partitioned into four fragments, and each fragment is replicated four times. After the bandwidth forecasts from the NWS are generated, the download tool then allocates threads so that if the forecasts are correct, and simultaneous connections execute at the forecasted bandwidth, each thread's download will complete at roughly the same time. Therefore, slower connections receive more threads. As in Test 1, the downloads are to a memory-mapped file in the client.

For the LAN version of the test, we used seven IBP depots on the UT computer science network, and the same client as in the previous tests. We then downloaded exNode files ranging from 1 MB to 512 MB using up to 16 threads. The results (Figure 9) are very similar to Test 1:

- The performance of one thread is worse than multiple threads.
- A small number of threads gives the best overall performance.
- The performance drops off once the file size exceeds the size of the client's memory.
- The limiting factor on many downloads is the performance of the networking hardware.

For WAN test, we stored all fragments and replicas on three depots at the University of California, San Diego. Each depot is a Linux machine with IBP depots limited to 10 threads each. Like the native IBP WAN tests, the multi-threaded exNode download tool saw consistent throughput gains by adding more threads up to a point. In these results, no more gains seem to be made after 24 threads (Figure 10).

This lack of increase above 24 threads may be due to the combination of our thread allocation policy and the number of threads running on the IBP depots. When selecting a depot from which to download a fragment, the tool chooses the copy with the best forecasted bandwidth. With this policy, even if another machine has nearly the same available bandwidth but its available bandwidth is 0.01 Mb/s less, the tool chooses the higher bandwidth always. Therefore, the policy does not take into account that the depots only had 10 threads to use, or that performance may be better when the load is spread evenly among depots with roughly the same performance. Therefore, this policy may be improved to show better performance.

3.4 Test 4 - Multi-Threaded Upload to Multiple Servers

Using our multi-threaded upload tool, we tested two different strategies for creating a replicated exNode in the WAN. The first strategy is to store the data directly from the client to the remote depots. The second strategy is to store a single copy to one of the remote depots and then use IBP's third-party copy to have the remote depot create the replicas. The goal of this strategy is to ease the performance bottleneck on client, and perhaps to take advantage of faster network paths between the IBP depots.

For each test, the goal was to create an exNode with three replicas of the data stored in multiple fragments on six machines in California (three at UCSD and three at UCSB). The client was the same as in the other tests.

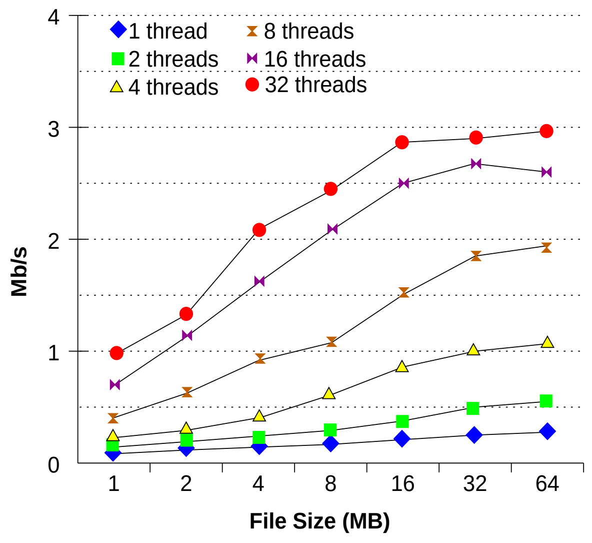

Figure 11: Multi-threaded upload to multiple servers on the WAN – all uploads performed by the client.

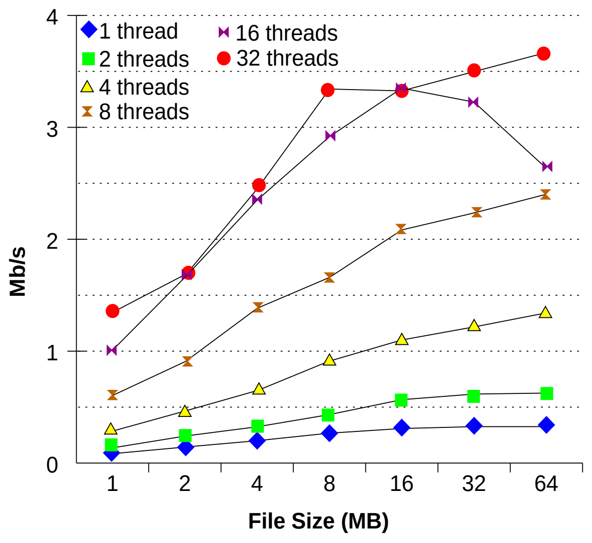

Figure 12: Multi-threaded upload to multiple servers on the WAN – one copy uploaded and then remotely stored to other depots.

The results are in Figures 11 and 12. Both show similar (albeit bad) performance, but the second policy outperforms the first in all cases. The best performance using the direct store method was just under 3 Mb/s while the throughput for the store and copy method was 3.7 Mb/s (Figures 11 and 12).

4 Discussion

The results presented above allow us to draw the following conclusions about multi-threading the Logistical Tools:

- On local-area connections, a small degree of multi-threading is most effective. A large number of simultaneous connections causes the performance of smaller transfers to suffer due to the fact that multiple TCP connections must be independently conditioned.
- However, on the LAN, multi-threading the tools allows them to reach the limits of the networking hardware.
- On the WAN, a large number of simultaneous connections is desirable, due to multiple network paths.
- IBP's decision to allow storage owners to limit the number of threads does indeed have performance implications on its users.

The two scheduling decisions addressed by this paper (the thread allocation downloading from multiple servers, and the upload/copy strategy as opposed to multiple uploads) were not explored fully enough to allow us to draw any definitive conclusions. However, the upload/copy strategy outperformed the multiple-upload strategy to a small degree in our tests. Further testing and experimentation will be necessary to draw more definitive conclusions.

5 Conclusions and Future Work

In this paper, we have reviewed the design of our Network Storage Stack, which provides abstractions and a methodology for applications to make use of storage as a network resource. Our experiments have allowed us to draw the conclusions summarized above about multi-threading in the Logistical Tools.

As stated above, the software for IBP, the L-Bone, the exNode and the Logistical Tools is publicly available and may be retrieved at `http://loci.cs.utk.edu`. The LoCI lab is especially interested in attracting more L-Bone participants, with the hope that the L-Bone can grow to over a petabyte of publicly accessible network storage in the next five years.

While the work detailed in this paper demonstrates the improvements in performance to the exNode tools, it also points out the need for additional research on scheduling strategies for uploads and downloads. We need to prevent under-utilization of threads if the depots are busy. We also would like to look into using far more depots across a much wider geographic area in hopes of improving performance even further.

To further improve fault-tolerance using the Logistical Tools, we intend to investigate the incorporation of coding blocks as supported entities in exNodes. For example, with parity coding blocks, we can equip the exNodes with the ability to use RAID techniques [CLG+94] to perform fault-tolerant downloads without requiring full replication. To reduce storage needs further, Reed-Solomon coding may be employed as well [Ber68, MS77, Pla97]. Finally, we also intend to add checksums as exNode metadata so that end-to-end guarantees may be made about the integrity of the data stored in IBP. All of these additions are future work for the LoCI lab.

Although not a performance or fault-tolerance issue, we will be adding more security features to the exNode and the Logistical Tools. Currently, the data stored in IBP depots are stored in the clear. In the future, exNodes will allow multiple types of encryption so that unencrypted data does not have to travel over the network, or be stored by IBP servers.

6 Related Work

In a previous paper [ASP+02], we showed that one result of this design is the ability to store data in a fault-tolerant way on the wide-area network. Preliminary performance results from that paper match the single-threaded performance results of this paper.

This work is different from work in distributed file systems (e.g. Coda [SKK+90], Jade [RP93] and Bayou [TTP+95]) in its freedom from the storage's underlying operating system (IBP works on Linux, Solaris, AIX, Mac OS X, and Windows), and its crossing of administrative domains. This work is also different from WebFS [VEA96]. Although WebFS allowed reading from the HTTP namespace which crosses administrative domains, it constrained writes to WebFS servers that run under one domain.

A project under development at the University of California, Berkeley, is OceanStore [KBC+00]. It is designed to be a global data storage utility. It does not rely on centralized state or control. Every write creates a new version of the file and all previous versions are retained. It also uses replication and error correction to provide fault-tolerance. This work differs from OceanStore in its layered approach for providing storage where each layer abstracts the layers beneath it to provide a flexible and scalable service.

The GridFTP project developed by the Globus group is adding extensions to traditional FTP. These extensions include using multiple streams for higher throughput, allowing third-part transfers and allowing for partial reads (i.e. downloading portions of the file) [GLO00].

Acknowledgments

The authors are grateful to Rich Wolski and Graziano Obertelli for granting access to the Santa Barbara machines, and Henri Casanova for granting access to the San Diego machines, and Jeremy Millar for administering the Texas depot. Additionally, the authors acknowledge Alex Bassi for authoring the exNode library along with Yong Zheng. Finally, the authors acknowledge Rich Wolski and Jack Dongarra for their vital participation in the Logistical Internetworking and Computing project.

References

[ASP+02] S. Atchley, S. Soltesz, J. S. Plank, M. Beck, and T. Moore. Fault-tolerance in the network storage stack. In *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, Ft. Lauderdale, FL, April 2002.

[Ber68] E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, New York, 1968.

[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, 2001.

[CLG+94] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[EPBW99] W. Elwasif, J. S. Plank, M. Beck, and R. Wolski. *IBP-Mail*: Controlled delivery of large mail files. In *NetStore '99: Network Storage Symposium*. Internet2, `http://dsi.internet2.edu/netstore99`, October 1999.

[GLO00] Gridftp - universal data transfer for the grid. Globus, `http://www.globus.org/datagrid/deliverables/C2WPdraft3.pdf`, September 2000.

[KBC$^+$00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2002)*. ASPLOS 2002, `http://www.csg.lcs.mit.edu/Users/rudolph/start.html`, November 2000.

[MS77] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes, Part I*. North-Holland Publishing Company, Amsterdam, New York, Oxford, 1977.

[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.

[PBE$^+$99] J. S. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the network. In *NetStore '99: Network Storage Symposium*. Internet2, `http://dsi.internet2.edu/netstore99`, October 1999.

[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.

[RP93] H. C. Rao and L. L. Peterson. Accessing files in an internet: The jade file system. *IEEE Transactions on Software Engineering*, 19(6), June 1993.

[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.

[SKK$^+$90] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990.

[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.

[TTP$^+$95] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *15th Symposium on Operating Systems Principles*, pages 172–183. ACM, December 1995.

[VEA96] A. Vahdat, P. Eastham, and T. Anderson. Webfs: A global cache coherent file system. Technical report, University of California, Berkeley, December 1996.

[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.