Logistical Networking Research and the Network Storage Stack

James S. Plank Micah Beck Terry Moore*

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

http://loci.cs.utk.edu
[plank,mbeck,tmoore]@cs.utk.edu

Introduction

Over the past few years, the **Lo**gistical **C**omputing and **I**nternetworking (**LoCI**) project at the University of Tennessee has been demonstrating the power of *Logistical Networking* in distributed and wide-area settings. Logistical Networking takes the rather unconventional view that storage can be used to augment data transmission as part of a unified network resource framework, rather than simply being a network-attached resource. The adjective "logistical" is meant to evoke an analogy with military and industrial networks for the movement of material which requires the coscheduling of long haul transportation, storage depots and local transportation as coordinate elements of a single infrastructure.

The motivation for Logistical Networking and demonstrations of its utility have been documented elsewhere [BMPS00, PBB+01]. This work-in-progress report presents a concept called the *Network Storage Stack*, and details the state of research centered around it.

Our goal with the Network Storage Stack (Figure 1) 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 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]. 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] for more detail on this approach).

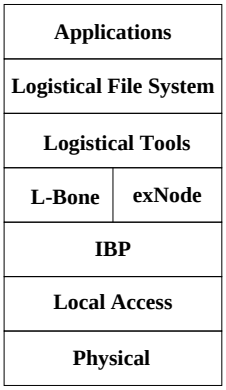

Figure 1: The Network Storage Stack

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 request an allocations 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

*This material is based upon work supported by the National Science Foundation under grants CCR-9703390, ACI-9876895, EIA-9975015, EIA-9972889, CCR-9703390, and Department of Energy under grant DE-FC0299ER25396 and the SciDAC/ASCR program, and the University of Tennessee Center for Information Technology Research.

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.

The 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 is 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 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. 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 stor-

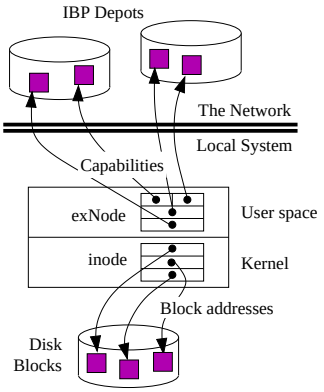

Figure 2: The exNode in comparison to the Unix inode

age 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. If the exNode is placed in a directory, the file it implements can be imbedded in a namespace. But if the exNode is sent as a mail attachment, there need not be a canonical location for it. The use of the exNode by varying applications provides interoperability similar to being attached to the same network file system. The exNode metadata is capable of expressing the following relationships between the file it implements and the storage resources that constitute the data component of the file's state:

- The portion of the file extent implemented by a particular resource (starting offset and ending offset in bytes).
- The service attributes of each constituent storage resource (e.g. reliability and performance metrics, duration)
- The total set of storage resources which implement the file and their aggregating function (e.g. simple union, parity storage scheme, more complex coding).

Logistical Tools

At the next level are tools that perform the actual aggregation of network storage resources, using the lower layers of the Network Stack. These tools take the form of client libraries that perform basic functionalities, and standalone programs built on top of the libraries. 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 for the upload. 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 that it represents 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). 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.

Refresh: This takes an exNode as input, and extends time limits of the IBP buffers that compose the file.

Augment: This takes an exNode as input, adds more replicas 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. Thus, **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.

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 tens 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 [Pla97]), downloads can be robust to even more complex failure scenarios.

Routing: As presented in the explanation of **trim** above, for the puposes of scheduling, or perhaps changing resource conditions, the tools may be used to move a file from one place to another.

The Logistical File System

While extremely powerful, the Logistical Tools still lack a level of abstraction that is necessary for truly effective use of network storage using our paradigm. That is, while they implement the *mechanism* for aggregating storage, they expose the *policy* of aggregation to higher layers. In other words, the tools make no decisions on how to replicate, how to partition, how to set time limits, etc. These decisions will be left to the highest layer, the Logistical File System (LoFS). This is a true file system built from time-limited network storage and the lower layers on the Network Storage Stack.

LoFS combines a Log-structured approach [OD89, RO91] with disk-array technology [CLG+94] so that read-write file operations may be implemented on top of append-only, faulty storage. Additionally, LoFS is responsible for:

- Maintaining enough replicas of data so that its files can absorb the loss of storage resources.
- Refreshing time limits.
- Scheduling the partitioning and replication of files to react to network and usage conditions, thereby achieving good downloading performance.
- Access control and protection.

With LoFS and the lower layers in place, it is our view that we will achieve our goals of inserting storage into the network resource fabric efficiently, flexibly, and scalably.

Current Status

This research project has been in place for several years, receiving funding from NSF, DOE, Internet2, and the state of Tennessee. All software and research papers may be obtained from the LoCI laboratory web page: `http://loci.cs.utk.edu`. IBP has been available for use for several years, and has shown its power in several research projects in many institutions. The L-Bone, exNode, and Logistical tools are available in alpha form, and were demonstrated at SC01 in an application called "IBPster," where an MP3 player played exNodes of MP3 files that were splattered throughout the country, by downloading them in a streaming fashion from their closest non-failed replicas in real time.

LoFS is still in its development stages. It will be based on the Swarm software from the University of Arizona [HMS99], which implements a log-structured file system on distributed platforms employing RAID coding for fault-tolerance.

References

[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.

[BMPS00] M. Beck, T. Moore, J. S. Plank, and M. Swany. Logistical networking: Sharing more than the wires. In C. A. Lee S. Hariri and C. S. Raghavendra, editors, *Active Middleware Services*. Kluwer Academic, Norwell, MA, 2000.

[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.

[HMS99] J. H. Hartman, I. Murdock, and T. Spalink. The Swarm scalable storage system. In *19th International Conference on Distributed Computing Systems (ICDCS)*, June 1999.

[OD89] J. K. Ousterhout and F. Douglis. Beating the I/O bottleneck: A case for log-structured file systems. *Operating Systems Review*, 23(1):11–27, January 1989.

[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.

[RO91] M. Rosenblum and J. K. Ousterhout. The design and implementation fo a log-structured file system. *Operating Systems Review*, 25(5):1–15, October 1991.

[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.

[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.

[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.