Logistical Storage Resources for the Grid

Alessandro Bassi[1], Micah Beck[1], Erika Fuentes[1], Terry Moore[1], James S. Plank[1]

[1]Logistical Computing and Internetworking Laboratory
Department of Computer Science
University of Tennessee
1122 Volunteer Blvd., Suite 203
Knoxville, TN 37996-3450
{abassi,mbeck,efuentes,tmoore,plank}@cs.utk.edu

Introduction

It is commonly observed that the continued exponential growth in the capacity of fundamental computing resources — processing power, communication bandwidth, and storage — is working a revolution in the capabilities and practices of the research community. It has become increasingly evident that the most revolutionary applications of this superabundance use *resource sharing* to enable new possibilities for collaboration and mutual benefit. Over the past 30 years, two basic models of resource sharing with different design goals have emerged. The differences between these two approaches, which we distinguish as the *Computer Center* and the *Internet* models, tend to generate divergent opportunity spaces, and it therefore becomes important to explore the alternative choices they present as we plan for and develop an information infrastructure for the scientific community in the next decade.

Interoperability and scalability are necessary design goals for distributed systems based on resource sharing, but the two models we consider differ in the positions they take along the continuum between total control and complete openness. That difference affects the tradeoffs they tend to make in fulfilling their other design goals. The Computer Center model, which came to maturity with the NSF Supercomputer Centers of the 80s and 90s, was developed in order to allow scarce and extremely valuable resources to be shared by a select community in an environment where security and accountability are major concerns. The form of sharing it implements is necessarily highly controlled – authentication and access control are its characteristic design issues. In the last few years this approach has given rise to a resource sharing paradigm known as information technology "Grids." Grids are designed to flexibly aggregate various types of highly distributed resources into unified platforms on which a wide range of "virtual organizations" can build. [11] By contrast, the Internet paradigm, which was developed over the same 30-year period, seeks to share network bandwidth for the purpose of universal communication among an international community of indefinite size. It uses lightweight allocation of network lins via packet routing in a public infrastructure to create a system that is designed to be open and easy to use, both in the sense of giving easy access to a basic set of network services and of allowing easy addition of privately provisioned resources to the public

infrastructure. While admission and accounting policies are difficult to implement in this model, the power of the universality and generality of the resource sharing it implements is undeniable.

Though experience with the Internet suggests that the transformative power of information technology is at its highest when the ease and openness of resource sharing is at its greatest, the Computer Center model is experiencing a rebirth in the Grid while the Internet paradigm has yet to be applied to any resource other than communication bandwidth. But we believe that the Internet model can be applied to other kinds of resources, and that, with the current Internet and the Web as a foundation, such an application can lead to similarly powerful results. The storage technology we have developed called the Internet Backplane Protocol (IBP) is designed to test this hypothesis and explore its implications. IBP is our primary tool in the study of logistical networking, a field motivated by viewing data transmission and storage within a unified framework. In this paper we explain the way in which IBP applies the Internet model to storage, describe the current API and the software that implements it, lay out the design issues which we are working to address, and finally characterize the future directions that this work will take.

Background: The Internet Protocol and the Internet Backplane Protocol

The Internet Backplane Protocol is a mechanism developed for the purpose of sharing storage resources across networks ranging from rack-mounted clusters in a single machine room to global networks. [4, 6, 15] To approximate the openness of the Internet paradigm for the case of storage, the design of IBP parallels key aspects of the design of IP, in particular IP datagram delivery. This service is based on packet delivery at the link level, but with more powerful and abstract features that allow it to scale globally. Its leading feature is the independence of IP datagrams from the attributes of the particular link layer, which is established as follows:

? Aggregation of link layer packets masks its limits on packet size;
? Fault detection with a single, simple failure model (faulty datagrams are dropped) masks the variety of different failure modes;
? Global addressing masks the difference between local area network addressing schemes and masks the local network's reconfiguration.

This higher level of abstraction allows a uniform IP model to be applied to network resources globally, and it is crucial to creating the most important difference between link layer packet delivery and IP datagram service. Namely,

> *Any participant in a routed IP network can make use of any link layer connection in the network regardless of who owns it. Routers aggregate individual link layer connections to create a global communication service.*

This IP-based aggregation of locally provisioned, link layer resources for the common purpose of universal connectivity constitutes the form of sharing that has made the Internet the foundation for a global information infrastructure.

IBP is designed to enable the sharing of storage resources within a community in much the same manner. Just as IP is a more abstract service based on link-layer datagram delivery, IBP is a more abstract service based on blocks of data (on disk, tape or other media) that are managed as "byte arrays." The independence of IBP byte arrays from the attributes of the particular *access layer* (which is our term for storage service at the local level) is established as follows:

? Aggregation of access layer blocks masks the fixed block size;

? Fault detection with a very simple failure model (faulty byte arrays are discarded) masks the variety of different failure modes;

? Global addressing based on global IP addresses masks the difference between access layer addressing schemes.

This higher level of abstraction allows a uniform IBP model to be applied to storage resources globally, and this is essential to creating the most important difference between access layer block storage and IBP byte array service:

> *Any participant in an IBP network can make use of any access layer storage resource in the network regardless of who owns it. The use of IP networking to access IBP storage resources creates a global storage service.*

Whatever the strengths of this application of the IP paradigm, however, it leads directly to two problems. First, in the case of storage, the chronic vulnerability of IP networks to Denial of Use (DoU) attacks is greatly amplified. The free sharing of communication within a routed IP network leaves every local network open to being overwhelmed by traffic from the wide area network, and consequently open to the unfortunate possibility of DoU from the network. While DoU attacks in the Internet can be detected and corrected, they cannot be effectively avoided. Yet this problem is not debilitating for two reasons: on the one hand, each datagram sent over a link uses only a tiny portion of the capacity of that link, so that DoU attacks require constant sending from multiple sources; on the other hand, monopolizing remote communication resources cannot profit the attacker in any way - except, of course, economic side-effects of attacking a competitor's resource. Unfortunately neither of these factors hold true for access layer storage resources. Once a data block is written to a storage medium, it occupies that portion of the medium until it is deallocated, so no constant sending is required. Moreover it is clear that monopolizing remote storage resources can be very profitable for an attacker and his applications.

The second problem with sharing storage network-style is that the usual definition of a storage service is based on processor-attached storage, and so it includes strong semantics (near-perfect reliability and availability) that are difficult to implement in the wide area network. Even in "storage area" or local area networks, these strong semantics can be difficult to implement and are a common cause of error conditions. When extended to the wide area, it becomes impossible to support such strong guarantees for storage access.

We have addressed both of these issues through special characteristics of the way IBP allocates storage:

? *Allocations of storage in IBP can be time limited.* When the lease on an allocation expires, the storage resource can be reused and all data structures associated with it can be deleted. An IBP allocation can be refused by a storage resource in response to over-allocation, much as routers can drop packets, and such "admission decisions" can be based on both size and duration. Forcing time limits puts transience into storage allocation, giving it some of the fluidity of datagram delivery.

? *The semantics of IBP storage allocation are weaker than the typical storage service*. Chosen to model storage accessed over the network, it is assumed that an IBP storage resource can be transiently unavailable. Since the user of remote storage resources is depending on so many uncontrolled remote variables, it may necessary to assume that storage can be permanently lost. Thus, *IBP is a "best effort" storage service*. To encourage the sharing of idle resources, IBP even supports "volatile" storage allocation semantics, where allocated storage can be revoked at any time. In all cases such weak semantics mean that the level of service must be characterized statistically.

Because of IBP's limitations on the size and duration of allocation and its weak allocation semantics, IBP does not directly implement reliable storage abstractions such as conventional files. Instead these must be built on top of IBP using techniques such as redundant storage, much as TCP builds on IP's unreliable datagram delivery in order to provide reliable transport.

The IBP Service, Client API, and Current Software

IBP storage resources are managed by "depots," or servers, on which clients perform remote storage operations. The IBP client calls fall into three different groups:

Table 1. IBP API Calls

Storage Management	Data Transfer	Depot Management
IBP_allocate IBP_manage	IBP_store IBP_load IBP_copy IBP_mcopy	IBP_status

`IBP_allocate` is used to allocate a byte array at an IBP depot, specifying the size, duration and other attributes. A chief design feature is the use of capabilities (cryptographically secure passwords). A successful `IBP_allocate` returns a set of three capabilities: one for reading, one for writing, and one for managing the allocated byte array. The `IBP_manage` call allows the client to manipulate the read and the write reference counter for a specific capability, probe the capability itself or change some of its characteristics. The `IBP_store` and `IBP_load` calls are two blocking calls that store and load the data on a particular capability; while with the `IBP_status` call it's possible to query a depot about its status and to change some

parameters. The `IBP_copy` and `IBP_mcopy` calls provide data transfer, and will be analyzed in more depth in section 4.

A more detailed account of the API and its other functions is available online (http://loci.cs.utk.edu/ibp/documents).

Prototype versions of IBP were available since 1999. Version 1.0 was released in March 2001, and the current release of the code (1.1.1), developed in C, is available for free download at our web site. It has been successfully tested under Linux, Solaris, AIX, DEC and Apple OS X machines. A Windows version also exists, for both client and depot, and a Java client library is also available.

The Data Mover

Since the primary intent of IBP is to provide a common abstraction of storage, it is arguable that third party transfer of data between depots is unnecessary. Indeed, it is logically possible to build an external service for moving data between depots that access IBP allocations using only the `IBP_load` and `IBP_store` calls; however, such a service would have to act as a proxy for clients, and this immediately raises trust and performance concerns. The `IBP_copy` and `IBP_mcopy` data movement calls were provided in order to allow a simple implementation that avoids these concerns, even if software architectures based on external data movement operations are still of great interest to us.

The intent of the basic `IBP_copy` call is to provide access to a simple data transfer over a TCP stream. This call is built in the IBP depot itself, to offer a simple solution for data movement; to achieve the transfer, the depot that receives the call is acting as a client doing an IBP_store on the target depot.

`IBP_mcopy` is a more general facility, designed to provide access to operations that range from simple variants of basic TCP-based data transfer to highly complex protocols using multicast and real-time adaptation of the transmission protocol, depending of the nature of the underlying backbone and of traffic concerns. In all cases, the caller has the capacity to determine the appropriateness of the operation to the depot's network environment, and to select what he believes the best data transfer strategy. A similar control is given over any error correction plan, should the data movement call return in failure.

The data mover is a plug-in module to an IBP depot that is activated either by an **`IBP_mcopy`** call or by an **`IBP_datamover`** call. The second call is not an API call, but an internal call made by the sending Data Mover. The sender depot is responsible for invoking a Data Mover plug-in on the receiving depot, and it accomplishes this by `fork`ing a data mover control process that sends an **`IBP_datamover`** request, causing the receiving depot to **`fork`** a symmetric data mover control. Sending and receiving control processes then **`exec`** the appropriate Data Mover plug-ins for the requested operation and these cooperate to perform the operation, then the plug-in at the sending depot replies to the client and then both plug-ins terminate.

The Data Mover software architecture can support a wide variety of operations, including:

? Point-to-multipoint through simple iterated TCP unicast transfers
? Point-to-multipoint through simultaneous threaded TCP unicast transfers.
? Unreliable UDP point-to-mulitpoint utilizing native IP mulitcast
? Reliable point-to-multipoint utilizing native IP multicast
? Fast, reliable UDP data transfer over private network links [5]

Experimental results

The design goal of the Data Mover plug-in and the function IBP_mcopy is to provide an optimized point to multipoint transfer tool, as well as a support for different protocols and methods for data transfer. In order to visualize and compare the behavior of the different methods to perform the data movement, two main experiments were completed under a specific environment where each of the nodes involved were interconnected with a stable, fast link within a local area network (LAN). The subsections 4.1.1 and 4.1.2 describe these experiments and their corresponding results, as well as a comparison of their performance and possible optimizations, using the Data Mover module, with the commonly used methods, such as IBP_copy and TCP respectively.

Point to Multipoint TCP

This subsection concentrates in comparing the transfer of different amounts of data using multiple simultaneous point to point TCP data transfers implemented at user level using threads, and using a single point to multipoint implementation of the Data Mover to transfer the same amounts of data.

As figure 2 reveals the latter approach shows an improvement in the overall transfer time. This experiment consisted of transferring several pieces of data of different sizes from one to various numbers of nodes.

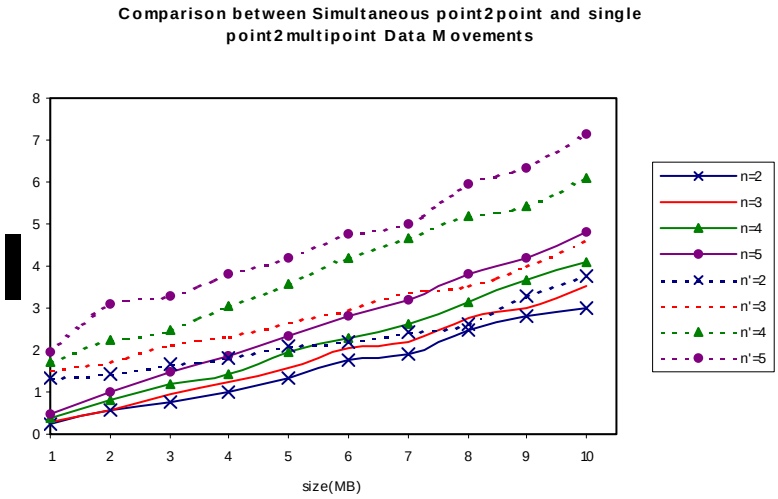

Fig. 2. Dotted lines represent the multiple simultaneous point to point TCP transfers the number of hosts used in different tests is given by n. The continuous lines represent point to multipoint approach, and number of hosts for this cases is given by n .

Point to Point UDP versus TCP

The experiment described in this subsection consists of a comparison between the transfer times using TCP and UDPBlaster point to point Data Movers.

Figure 3 shows the improvement achieved by using UDPBlaster. The Data Mover plug-in could support a variety of protocols and methods, however, for the purpose of this experiment we concentrate on the comparison of TCP versus UDP, to show how the improvement can be achieved within the same data mover using different protocol depending on the characteristics of the backbone being used. It is important to note that since this protocol is still in the experimental phase may behave differently in diverse test environments under different circumstances.

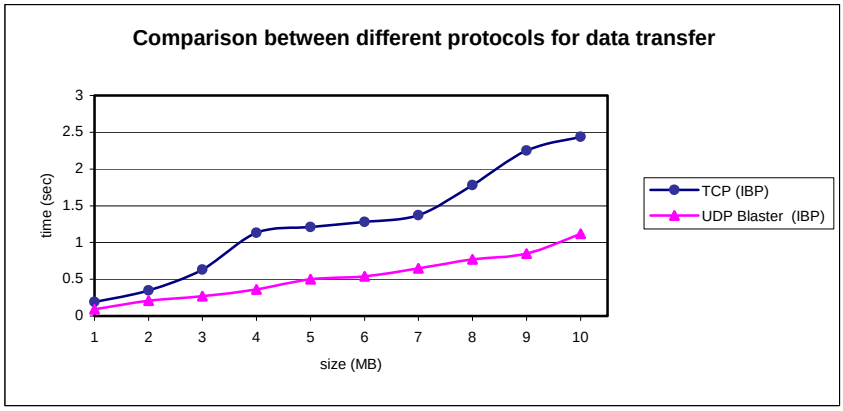

Fig. 3. Improvement seen by using UDPBlaster

The exNode

One of the key strategies of the IBP project was adhering to a very simple philosophy: IBP models access layer storage resources as closely as possible while still maintaining the ability to share those resources between applications While this gives rise to a very general service, it results in semantics that might be too weak to be conveniently used directly by many applications. As in the networking field the IP protocol alone does not guarantee many highly desirable characteristics, and needs to be complemented by a transport protocol such as TCP, what is needed is a service at the next layer that, working in synergy with IBP, implements stronger allocation properties such as reliability and arbitrary duration that IBP does not support.

Following the example of the Unix inode, that aggregates disk blocks to implement a file, we have chosen to implement a single generalized data structure, which we call an *external node*, or *exNode*, in order to manage aggregate allocations that can be used in implementing network storage with many different strong semantic properties. Rather than aggregating blocks on a single disk volume, the exNode aggregates storage allocations on the Internet, and the exposed nature of IBP makes IBP byte-arrays exceptionally well adapted to such aggregations. In the present context the key point about the design of the exNode is that it has allowed us to create an abstraction of a network file to layer over IBP-based storage in a way that is completely consistent with the exposed resource approach.

The exNode library has a set of calls that allow an application to create and destroy an exNode, to add or delete a mapping from it, to query it with a set of criteria, and to produce an XML serialization, to allow the use of XML-based tools and interoperability with XML systems and protocols.

Table 2. exNode API calls

ExNode Management	**Segment Management**
XndCreateExNode xndFreeExNode	xndCreateSegment xndFreeSegment xndAppendSegment xndDeleteSegment
Segment Query	**ExNode Serialization**
XndQuery xndEnumNext xndFreeEnum	XndSerialize xndDeserialize

A more complete explanation of the exNode and the tools we developed is available online (http://loci.cs.utk.edu/exNode/index.htm). The software library, written in c, is currently in its beta release, while a java version is under development. We plan to make them available together with our IBP distribution in the very near future through our web site.

Experiences and Applications

Our method in developing the Internet Backplane Protocol is based on implementation and experimentation. A number of simple applications from within our own research group have formed the basis of the experience that has guided our work. In addition, a few external application groups have picked up on early implementations of IBP and contribute by both tracking our releases and giving us needed feedback for future developments. However, it is only with the upcoming release of the exNode library and serialization that we believe a wide application community will find the supporting tools necessary to adopt an IBP-based methodology.

? IBP-mail is a system that uses IBP to transmit and deliver mail attachments that require storage resources beyond the capacity of standard mail servers[10]. To use IBP-Mail, the sender first uploads the attachment into a suitable IBP server and then forwards the read capability to the receiver, who can use it to download the attachment. An initial version of IBP-mail used custom CGI scripts to chose the depot and implement the upload and download functions. The capability was transmitted between sender and receiver embedded in a form that invoked the receiving CGI. This custom architecture has been replaced by a generic mechanism for storing IBP capabilities (see section 6 on the exNode) which allows them to be sent between sender and receiver as a simple file attachment in a standard serialized format. The manipulation of files stored in IBP format is now handled through generic tools operating on the serialized exNode data structure, and so today IBP-Mail is not so much an application as a way of using those standard tools together with the standard MIME attachment facility in e-mail.

? NetSolve is a distributed computation tool created by Casanova and Dongarra [8] to provide remote invocation of numerical libraries for scientific code. One

shortcoming of the initial NetSolve architecture is that it is stateless: a series of calls to the same server cannot, for instance, cache arguments or results in order to avoid unnecessary data movement in subsequent calls. One of the approaches taken to optimize NetSolve was to use an IBP depot at the server to implement a cache under the control of the client. Such short-lived use of IBP for value caching is able to make good use of even volatile and time-limited allocations [2]. Experiments conducted by the NetSolve group, using a Client application at the University of California, San Diego requesting a pool of computational resources and Data Storage at the University of Tennessee, showed a much-improved efficiency. [1, 2, 9].

? TAMANOIR [12] is a project developed by the RESAM lab of the Ecole Normale Superieure of Lyon in the field of Active Networking. It is a framework that allows users to easily deploy and maintain distributed active routers in a wide area network. IBP depots will be among the standard tools available to services implemented within the TAMANOIR framework, along with other basic tools such as the routing manager and stream monitoring tools. It will also be used to implement distribution and caching of services (distributed as Java byte-code modules) that are loaded by TAMANOIR on-demand, freeing TAMANOIR to manage only is own internal cache of services.

Related Work

IBP occupies an architectural niche similar to that of network file systems such as AFS [14] and Network Attached Storage appliances [13], but its model of storage is more primitive, making it similar in some ways to Storage Area Networking (SAN) technologies developed for local networks. In the Grid community, projects such as GASS [7] and the SDSC Storage Resource Broker [3] are file system overlays that implement a uniform file access interface and also impose uniform directory, authentication and access control frameworks on their users.

Conclusions

While some ways of engineering for resource sharing, such as the *Computer Center model*, focus on optimizing the use of scarce resources within selected communities, the exponential growth in all areas of computing resources has created the opportunity to explore a different problem, viz. designing new architectures that can take more meaningful advantage of this bounty. The approach presented in this paper is based on the *Internet model of resource sharing* and represents one general way of using the rising flood of storage resources to create a common distributed infrastructure that can share the growing surplus of storage in a way analogous to the way the current network shares communication bandwidth. It uses the Internet Backplane Protocol (IBP), which is designed on the model of IP, to allow storage resources to be shared

by users and applications in a way that is as open and as easy to use as possible while maintaining a necessary minimum of security and protection from abuse. IPB lays the foundation for the intermediate resource management components, accessible to every end-system, which must be introduced to govern the way that applications access, drw from, and utilize this common pool in a fully storage-enabled Internet.

Bibliography

1. D. C. Arnold, D. Bachmannd, and J. Dongarra, "Request Sequencing: Optimizing Communication for the Grid," in *Euro-Par 2000 -- Parallel Processing, 6th International Euro-Par Conference*, vol. 1900, *Lecture Notes in Computer Science*, A. Bode, T. Ludwig, W. Karl, and R. Wismuller, Eds. Munich: Springer Verlag, 2000, pp. 1213-1222.
2. D. C. Arnold, S. S. Vahdiyar, and J. Dongarra, "On the Convergence of Computational and Data Grids," *Parallel Processing Letters*, vol. 11, no. 2, pp. 187-202, June/September, 2001.
3. C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC Storage Resource Broker," presented at CASCON'98, Toronto, Canada, 1998.
4. A. Bassi, M. Beck, J. Plank, and R. Wolski, "Internet Backplane Protocol: API 1.0," Department of Computer Science, University of Tennessee, Knoxville, TN, CS Technical Report, ut-cs-01-455, March 16, 2001. http://www.cs.utk.edu/~library/2001.html.
5. M. Beck and E. Fuentes, "A UDP-Based Protocol for Fast File Transfer," Department of Computer Science, University of Tennessee, Knoxville, TN, CS Technical Report, ut-cs-01-456, June, 2001. http://www.cs.utk.edu/~library/2001.html.
6. M. Beck, T. Moore, J. Plank, and M. Swany, "Logistical Networking: Sharing More Than the Wires," in *Active Middleware Services*, vol. 583, *The Kluwer International Series in Engineering and Computer Science*, S. Hariri, C. Lee, and C. Raghavendra, Eds. Boston: Kluwer Academic Publishers, 2000.
7. J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," presented at Sixth Workshop on I/O in Parallel and Distributed Systems, May 5, 1999, 1999.
8. H. Casanova and J. Dongarra, "Applying NetSolve's Network Enabled Server," *IEEE Computational Science & Engineering*, vol. 5, no. 3, pp. 57-661998.
9. H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," presented at 9th Heterogeneous Computing Workshop (HCW'00), May 2000, 2000.
10. W. Elwasif, J. Plank, M. Beck, and R. Wolski, "IBP-Mail: Controlled Delivery of Large Mail Files," presented at NetStore99: The Network Storage Symposium, Seattle, WA, 1999.
11. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of SuperComputer Applications*, vol. 15, no. 3, To appear, 2001.
12. J. Gelas and L. Lefevre, "TAMANOIR : A High Performance Active Network Framework," in *Active Middleware Services*, vol. 583, *The Kluwer International Series in Engineering and Computer Science*, S. Hariri, C. Lee, and C. Raghavendra, Eds. Boston: Kluwer Academic Publishers, 2000.
13. G. Gibson and R. V. Meter, "Network Attached Storage Architecture," *Communications of the ACM*, vol. 43, no. 11, pp. 37-45, November, 2000.

14. J. H. Morris, M. Satyanarayan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, and F. D. Smith, "Andrew: A Distributed Personal Computing Environment," *Communications of the ACM*, vol. 29, no. 3, pp. 184-201, March, 1986.
15. J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski, "The Internet Backplane Protocol: Storage in the Network," presented at NetStore99: The Network Storage Symposium, Seattle, WA, 1999.