

Optimizing Performance and Enhancing Functionality of Distributed Applications Using Logistical Networking

Submitted by: Micah Beck
1122 Volunteer Blvd., Suite 203
Knoxville, TN 37996-3450

mbeck@cs.utk.edu
Phone: 865-974-3548
Fax: 865-874-8296

Project Summary

There is wide agreement that an essential key to the new era of simulation-intensive, collaboration-enabled scientific computing, envisioned by DOE's *Scientific Discovery through Advanced Computing (SciDAC)* initiative, is the development of advanced network and middleware services that can provide reliable, fast, flexible, scalable, and efficient delivery of data to support distributed and high performance applications of all types. The research on *Logistical Networking* that we propose represents an aggressive approach that aims to extract maximum leverage to achieve this end from the ongoing, exponential growth in all types of computing resources — processing power, communication bandwidth, and storage.

Logistical Networking is the global scheduling and optimization of data movement, storage and computation based on a model that takes into account **all** of the network's underlying physical resources, especially storage. By adding a uniform model of storage to the definition of the network and exposing it for direct scheduling, Logistical Networking will enable SciDAC application designers to make much stronger assumptions about the ability of next generation applications to manage distributed state and engage in asynchronous communications of all types. We believe that sharing storage resources in this way is essential to robust collaboration based on data sharing that SciDAC envisions. The research proposed here will allow us to

- perfect the engineering of the necessary middleware for remote storage management that Logistical Networking requires, making it fast, robust, scalable, and easy to use,
- experiment with and explore the capabilities of this middleware in a range of innovative applications with high performance and/or unique functionality; and
- disseminate Logistical Networking by deploying these technologies and applications to a broad community of users on a special testbed made available for this purpose.

We have designed and implemented fundamental, low level middleware, called the *Internet Backplane Protocol (IBP)* for managing remote storage to support Logistical Networking. With a stable, production quality implementation of IBP in hand and significant experience using it successfully in a variety of experiments and applications, we are prepared not only to push our research into promising new areas, but also develop the software infrastructure necessary to build the *storage-enabled Internet* and make Logistical Networking ubiquitous within the SciDAC community. This work falls into three main parts.

First, we will develop the complementary middleware components necessary to make IBP more robust, scalable, and easy to use. The technologies include the *exNode*, which is an XML-encoded data structure for building a strong file abstraction on the weak semantic properties of IBP, and a *Data Mover Interface* for optimizing transfers between IBP "depots." This effort will lay an adequate foundation for large-scale deployment of storage-enabled internetworking across the national research community.

Second, with these software tools in hand, we will explore fertile new areas and applications of research. This will include investigations of the generalized logistical routing problem, Logistical Session Layer technology for performance optimization and dynamic overlay networks, application-independent caching services for state management in computationally intensive network applications, and universal IBP-mail that supports e-mail attachments of *arbitrary* size for distributed scientific collaboration.

Finally, we will continue disseminate this technology via a research testbed called the *Logistical Backbone (L-Bone)*. The L-Bone consists of IBP storage depots installed on the nodes of the Internet 2 Distributed Storage Infrastructure (I2-DSI) and additional depots managed by our academic and corporate research partners. This will put Logistical Networking technology directly into the hands of the research community accessible to Internet2, ESNet, and other advanced networks.

1. Introduction: Logistical Networking for Advanced Scientific Computing

While explosive growth in the use of compute and data intensive simulation continues to transform the practice of scientific investigation across every field, a parallel transformation is also revolutionizing the ability of researchers to collaborate across geographic and disciplinary barriers. The vision of a new era of science, produced by the convergence and maturation of these two powerful trends in scientific computing, is shared broadly within the research community and forms the foundation of the Department of the Energy's (DOE) *Scientific Discovery through Advanced Computing (SciDAC)* initiative. But as the background studies that inform SciDAC [1-3] make clear, an indispensable key to realizing this vision is the development of advanced network and middleware services that can provide reliable, fast, flexible, scalable, and cost-effective delivery of data to support distributed and high performance applications of all types. The research we are engaged in under the *Logistical Computing and Internetworking (LoCI)* project at the University of Tennessee aims to respond to this challenge with a simple but innovative strategy.

At the base of the LoCI project is a richer view of the use of storage in communication. Most current approaches to advanced network services rely on the standard end-to-end model of communication [4] where the state of network flows is to be maintained at the end nodes and not in the network. On this view, the network itself offers no model of storage; so it is not surprising that there is no current service that makes storage available for general use "in the network", i.e. available in the sense that it is easily and flexibly usable by a broad community without individual authorization or per-transaction billing. *There is no shared distributed storage infrastructure for the support of asynchronous communication generally.*

By contrast, our concept of *Logistical Networking* [5, 6] represents a more radical approach, an approach that tries to get maximum leverage out of the ongoing, exponential growth in all types of computing resources — processing power, communication bandwidth, and storage:

*Logistical Networking is the global scheduling and optimization of data movement, storage and computation based on a model that takes into account **all** the network's underlying physical resources, including storage and computation.*

By adding a uniform model of storage to the definition of the network, and thereby exposing such embedded storage for direct scheduling, we intend to make it possible for SciDAC application designers to make much stronger assumptions about the ability of next generation applications to manage distributed state and engage in asynchronous communications of all types. *Logistical Networking* offers a general way of using the rising flood of computing resources to create a common distributed storage infrastructure that can share out the rapidly growing bounty of storage (and computation) the way the current network shares out bandwidth for synchronous communication (i.e. the sender and receiver are simultaneously connected to the network.)

The goal of the research effort we propose below is to develop the *Logistical Networking* middleware to support the creation of a *storage-enabled Internet*, whose anticipated benefits include the ability to

- Provision a ubiquitous substrate of network storage for the general support of asynchronous connectivity, fostering performance enhancements for collaborative applications individually and greatly facilitating the interoperability that will be essential to all SciDAC's collaborative applications collectively.
- Enhance content distribution services with both client and server-driven replication strategies (including, but not limited to caching, content push, multicast support, replica management) for improved performance.
- Provide broad-based support that enables advanced application schedulers to include the staging of data near the processing resources for better resource utilization and better performance.
- Deliver enhanced management of communication state in the network that can allow some applications to implement Quality of Service (QoS) guarantees without the need for end-to-end reservation or bandwidth brokering in the style of the ReSource reservation Protocol (RSVP) [7, 8] or Differentiated Services [9], support applications that are currently impossible due to resource

constraints or the lower bound on latency imposed by the speed of light, and facilitate management of the load of QoS traffic on the backbone.

- Create a ubiquitous foundation for achieving fault-tolerance in advanced network applications.

A common reaction to this idea of a storage-enabled Internet is that many of these functions could be served by using individually owned resources accessed by various protocols already established. Examples are mail and news servers, Web caches, and distributed file and database systems. While it is true that each of these cases works without the use of common storage resources, the result is a balkanization of resources dedicated to specific applications and a lack of interoperability between similar applications. Today, objects shared during collaboration are held in private storage. But this approach gives out when, for instance, an average user decides to bring a data set into a collaborative session that is so large it cannot be held in private storage owned by the participants in the session, or when different collaborative applications fail to interoperate because there is no private storage that is jointly accessible to them at the time required. In that case, the community may be willing to provision massive storage for time-limited use and to make it freely available. Although the community is unlikely to take such a step for a particular application, it may be willing to do so if, as in the case of provisioning IP networks, all users can share it for a wide range of applications. High degrees of interoperability tend to enable ambitious community investment.

Below we propose research designed to create the middleware necessary for the storage-enabled Internet and further explore the value of Logistical Networking as a foundation for SciDAC's collaborative applications. Section 2 describes the background and motivation for our focus on Logistical Networking, including the LoCI hypothesis on which it depends and the basic mechanism, called the Internet Backplane Protocol (IBP) [5, 6], which we have implemented to experiment with it. Since IBP software has now been implemented in production quality form, Section 3 presents the additional development work — including a file abstraction for storage in the wide area, called the *exNode* — which is necessary to build in the robustness, ease of use, and scalability that the storage-enabled Internet will require. In section 4 we describe the key research issues for Logistical Networking that we intend to investigate, including some applications that have driven or will drive our work. Since our research approach focuses on tools that model network resources in ways that can be used by real application communities, we view the key issues and the prototype applications as complementary aspects of our research agenda. For that same reason, the deployment and use of these applications on our Logistical Networking testbed, which we describe in Section 5, is a key element for the success of our research plan. After describing related work in Section 6, we conclude in Section 7 by describing our plan in more detail, including a table of milestones for our experimental software.

2. Background and Motivation for LoCI Research

Achieving robust functionality and performance in loosely coupled, network computing environments has always been difficult. The LoCI project explores a hypothesis about how to address these difficulties in order to achieve better resource utilization, performance, and functionality for distributed applications. LoCI's basic hypothesis is this:

- If**
- 1) network resources (viz. distributed storage, communication and computation) are predictably allocated, and
 - 2) these resources can be flexibly coscheduled,

Then advanced applications can be implemented with higher performance, increased functionality, and/or more efficient overall use of communication, computational, and storage resources.

As this formulation makes clear, LoCI revolves around the management, monitoring and co-scheduling of communication, storage and processing:

- **Communication:** managing, monitoring and scheduling communication is straightforward. Either an end-to-end QoS facility like RSVP [7] may be used to schedule point-to-point communication, or a monitoring and prediction facility like the Network Weather Service (NWS) [10] may be used to schedule best-effort networks in a reasonably effective manner.

- **Storage:** no standard systems and interfaces exist for the management of shared remote storage. HTTP and FTP servers offer control of remote readable storage, and distributed file systems are effective when all participating nodes agree on an administrative domain. However, none of these serve LoCI's need for a system that allocates storage according to a scheme that exposes the resource for *shared* scheduling by network applications. For this reason, we have designed and implemented a piece of middleware called the *Internet Backplane Protocol (IBP)*. IBP will be detailed in sections 2.2-3. For monitoring and prediction, IBP will employ Network Weather Service (NWS) primitives.
- **Processing:** not all parts of the LoCI project involve the direct scheduling and management of remote processors. The parts that now make use of current systems (e.g NetSolve [11], Globus [12], and NWS [10]) to perform processor allocation, monitoring and scheduling.

What makes the LoCI approach different from the status quo in network computing systems is the aggressive way it exposes resources *other than* synchronous communication bandwidth for applications to use. This is exemplified in our current research, described below, which focuses on the use of network storage. By using our simple protocol, *IPB* [5, 13], we can add explicit buffer and state management into network applications in a very flexible, scalable, and interoperable way. From the performance point of view, the reason for this is sound. In the absence of remote storage management, the only way to schedule data movement from point A to point B is to make an end-to-end QoS reservation, or to successfully predict the observed bandwidth between A and B. By providing a shared storage substrate for general use, any application can schedule data movement on shorter links that are easier to observe and predict, thereby getting more effective resource usage and higher performance.

2.1 An Illustration of Logistical Networking

The term “collaboratory” was coined by William Wulf, who identified its collaborative aspect as being comprised of “interaction with colleagues, accessing instrumentation, *sharing data and computational resources*, and accessing information in digital libraries”[14]. It is our belief that the dependence of collaborative technology on end-to-end networking has led to the emphasis of *mutual access* to private resources and the neglect of *resource sharing*. To see more concretely how the kind of collaboration enabled by Logistical Networking differs from the typical approach, consider an instance of distributed collaboration that is likely to become commonplace with the realization of the SciDAC vision. Two scientists, Dr. S and Dr. J, who work at different geographical locations, want to meet to discuss an ongoing, simulation-intensive research project and plan their next steps. When we examine the problem of providing the necessary communication infrastructure for such a scenario, we can distinguish between two different components of the scientist’s communication. Messages that do not exist before they are sent are considered part of the synchronous *CONVERSATION*. Conversation includes live audio and video streams, as well as chat and synchronous document sharing. On the other hand a message that exists substantially before it is sent is considered to be *CONTEXT*; context can include data sets, images, recordings of previous simulation runs, and in general any other materials that exist before the meeting starts. With respect to QoS, these two components of the overall communication can be handled very differently:

- Delivery of contextual data can be staged asynchronously — Since the contextual data exists prior to the communication, the data can be moved in advance from an end node to some intermediate point (which we call a *data depot*) and stored there in a data staging step. Then the data is delivered from the depot to the receiver with the required level of service. Both movements of the data will require network resources, but the data staging step will be able to use best effort, class of service, or reservation at very low levels. The final delivery step does have to meet the QoS specified by the receiver and may use standard end-to-end QoS. But this second operation can be performed in a variety of ways, e.g. through an over-provisioned local area network, across a single switch [15], or within a local ATM cloud. This is QoS through Logistical Networking.
- Delivery of conversational data requires end-to-end QoS — Data cannot be staged when it either has to be received in real time as it is created or when the decision to send it cannot be anticipated in advance. Conversation, in our generic sense, is made up of both these types of data. The typical

approach to achieving QoS for these cases is *end-to-end reservation* [7, 16]. The network resources necessary to send the data with required QoS are declared in advance by the respective senders and guaranteed by the network to be available.

The difference between Logistical Networking and the end-to-end approach is illustrated in Figure 1. Here, **A** is the sender, **B** is the receiver, and **C** represents a data depot. As the diagram suggests, **A** is relatively "far away" from **B** (e.g. across the national backbone), while **C** is much "closer" (e.g. on the same LAN). The figure compares the two forms of data delivery for a simple case in which a communication is scheduled at time **0** to occur at time **t**. Standard QoS (dashed red line) moves **all** the data components of the communication in a single step. By contrast Logistical Networking (dotted blue line) uses a data staging step and a final delivery step. For those data elements that can be staged, the result at the receiving end can be just the same quality as end-to-end delivery in one step, but the work of the sender and the network has been spread over time and storage space.

These two approaches are both compatible and complementary, as is evident from the fact that the delivery step of a logistical communication may use standard QoS. Yet some performance requirements that are impossible to achieve through the standard end-to-end approach alone may be possible with logistical communication. For instance, take any case in which latency requirements are lower than the limit imposed by the speed of light, as certainly may be the case for some high-performance distributed applications under SciDAC. Logistical Networking can bring the data closer to the receiver and push perceived latency below this limit, whereas end-to-end QoS, which treats communication as if it was all "conversation" and no "context," is limited by the physics of real-time interaction.

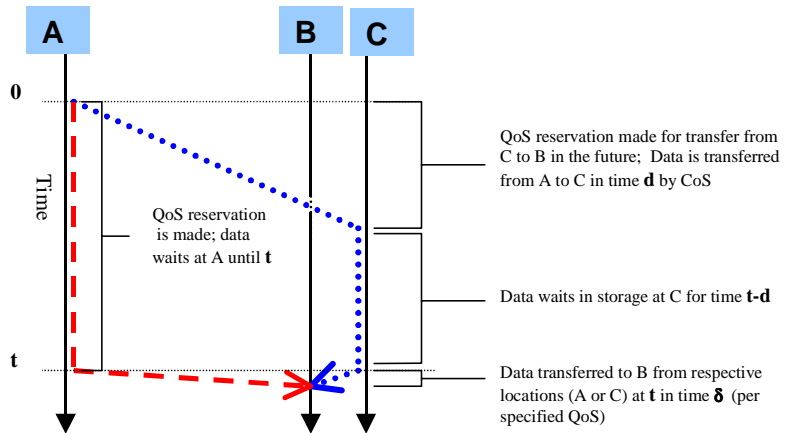


Figure 1: Scheduled delivery of data via standard QoS (dashed line) and Logistical Networking (dotted line)

Moreover, since sharing context in this sense is fundamental to collaboration, a storage-enabled Internet that is well provisioned for Logistical Networking would form a powerful complement to the synchronous IP network in supporting interoperability, scalability, and robustness for SciDAC's collaborative applications. For example, in the current network, Dr. S and Dr. J can only share large amounts of context by acquiring sufficient private space to which both can be given access. If the size of the data they want to share, however, happens to outstrip the size of this private space, or if the simulation they want to jointly view cannot easily dump its output there, or if they want to use several applications in tandem and these applications cannot be given joint access to this private space, then this lack of interoperability will cause their collaboration to suffer. In a storage-enabled Internet, on the other hand, there is a substrate of shared storage resources that the developers of collaborative applications can take for granted in asynchronous communication of all kinds, just as they now take IP connectivity for granted in synchronous communication. In such a network, every user and application would be able to share context whenever necessary through the same standard, application independent interfaces. Without the resource sharing inherent in Logistical Networking, SciDAC collaborations may be limited to mutual remote access, and the promise of working together through resource sharing may remain forever unfulfilled.

2.2 The Internet Backplane Protocol (IBP)

IBP is middleware for managing and using remote storage [5, 13]. Invented as part of LoCI to support Logistical Networking in large scale, distributed systems and applications, it acquired its name because it

was designed to enable applications to treat the Internet as if it were a processor backplane. Whereas on a typical backplane, the user has access to memory and peripherals and can direct communication between them with DMA, IBP gives the user access to remote storage and standard Internet resources (e.g. content servers implemented with standard sockets) and can direct communication between them with the IBP API.

By providing a uniform, application-independent interface to storage in the network, IBP makes it possible for applications of all kinds to use Logistical Networking to exploit data locality and more effectively manage buffer resources. We believe it represents the kind of middleware needed to overcome the current balkanization of state management capabilities on the Internet. IBP will allow *any* application that needs to manage distributed state to benefit from the kind of standardization, interoperability, and scalability that have made the Internet into such a powerful communication tool.

Since IBP draws on elements from different traditional designs, it does not fit comfortably into the usual categories of mechanisms for state management: IBP can be viewed as a mechanism to manage either *communication buffers* or *remote files*. Both characterizations are equally valid and useful in different situations. If, in order to use a neutral terminology, we simply refer to the units of data that IBP manages as *byte arrays*, then these different views of IBP can be presented as follows:

- **IBP as buffer management** — Communication between nodes on the Internet is built upon the basic operation of delivering packets from sender to receiver, where each packet is *buffered* at intermediate nodes. Because the capacity of even large storage systems is tiny compared with the amount of data that flows through the Internet, allocation of communication buffers must be time limited. In current routers and switches, time-limited allocation is implemented by use of FIFO buffers, serviced under the constraints of fair queuing. Against this background, *IBP byte arrays can be viewed as application-managed communication buffers in the network*. IBP supports time-limited allocation and FIFO disciplines to constrain the use of storage. With such constraints in place, applications that use these buffers can improve communication and network utilization by way of application-driven staging and course-grained routing of data.
- **IBP as file management** — Since high-end Internet applications often transfer gigabytes of data, the systems to manage storage resources for such applications are often on the scale of gigabytes or terabytes in size. Storage on this scale is usually managed using highly structured file systems or databases with complex naming, protection, and robustness semantics. Normally such storage resources are treated as part of a host system and therefore as more or less private. From this point of view *IBP byte arrays can be viewed as files that live in the network*. IBP allows an application to read and write data stored at remote sites, as well as direct the movement of data among storage sites and to multiple receivers. In this way IBP creates a network of shareable storage in the same way that standard networks provide shareable bandwidth for file transfer.

This characterization of IBP as a mechanism for managing state in the network supplies an operational understanding of our approach to the problem of Logistical Networking for storage. The usual view is that routing of packets through a network is a series of *spatial* choices that allows control of only one aspect of data movement. An incoming packet is sent out on one of several alternative links, but any particular packet is held in communication buffers for as short a time as possible. But Logistical Networking with storage makes it possible to route packets in two dimensions, not just one: IBP allows for data to be stored at one location while *en route* from sender to receiver, adding the ability to control data movement *temporally* as well as spatially. This is a key aspect of Logistical Networking, but to see how IBP implements this concept we need to examine its API in detail.

2.2.1 IBP Structure and Client API

IBP has been designed to be a minimal abstraction of storage to serve the needs of Logistical Networking. Its fundamental operations are:

1. Allocating a byte array for storing data.
2. Moving data from a sender to a byte array.

3. Delivering data from a byte array to a receiver (either another byte array or a client).

We have defined and implemented a client API for IBP that consists of seven procedure calls and server daemon software that makes local storage available for remote management. Connections between clients and servers are made through TCP/IP sockets, but we are testing the integration of other network protocols (i.e. UDP) and various other means to improve the communication performance as much as possible.

IBP client calls may be made by *anyone* who can attach to an IBP server (which we also call an *IBP depot* to emphasize its logistical functionality). IBP depots require only storage and networking resources, and running one does not necessarily require supervisory privileges. These servers implement policies that allow an initiating user some control over how IBP makes use of storage. An IBP server may be restricted to use only idle physical memory and disk resources, or to enforce a time-limit on all allocations, ensuring that the host machine is either not impacted or that encourage users to experiment with logistical networking without over-committing server resources.

Logically speaking, the IBP client sees a depot's storage resources as a collection of append-only byte arrays. There are no directory structures or client-assigned file names. Clients initially gain access to byte arrays by allocating storage on an IBP server. If the allocation is successful, the server returns three *capabilities* to the client: one for reading, one for writing, and one for management. These capabilities can be viewed as names that are assigned by the server. Currently, each capability is a text string encoded with the IP identity of the IBP server, plus other information to be interpreted only by the server. This approach enables applications to pass IBP capabilities among themselves without registering these operations with IBP, thus supporting high-performance without sacrificing the correctness of applications.

The IBP client API consists of seven procedure calls, broken into three groups, as shown in Table 1 below. For clarity, we omit error handling and security considerations. Each call does include a timeout so that network failures may be tolerated gracefully. The full API is described separately [Bassi, 2001 #330] and is available at <http://icl.cs.utk.edu/ibp>.

Table 1: IBP Client Calls

Allocation	Reading/Writing	Management
IBP_allocate	IBP_store, IBP_load, IBP_copy, IBP_mcopy	IBP_manage, IBP_status

The heart of IBP's innovative storage model is its approach to allocation. Storage resources that are part of the network, as logistical networking intends them to be, cannot be allocated in the same way as they are on a host system. To understand how IBP needs to treat allocation of storage for the purposes of logistical networking, it is helpful to consider the problem of sharing resources in the Internet, and how that situation compares with the allocation of storage on host systems.

In the Internet, the basic shared resources are data transmission and routing. The greatest impediment to sharing these resources is the risk that their owners will be denied the use of them. The reason that the Internet can function in the face of the possibility of denial-of-use attacks is that it is not possible for the attacker to profit in proportion to their own effort, expense and risk. When other resources, such as disk space in spool directories, are shared, we tend to find administrative mechanisms that limit their use by restricting either the size of allocations or the amount of time for which data will be held.

By contrast, a user of a host storage system is usually an authenticated member of some community that has the right to allocate certain resources and to use them indefinitely. Consequently, sharing of resources allocated in this way cannot extend to an arbitrary community. For example, an anonymous FTP server with open write permissions is an invitation for someone to monopolize those resources; such servers must be allowed to delete stored material at will.

In order to make it possible to treat storage as a shared network resource, IBP supports some of these administrative limits on allocation, while at the same time seeking to provide guarantees for the client that

are as strong as possible. So, for example, under IBP allocation can be restricted to a certain length of time, or specified in a way that permits the server to revoke the allocation at will. Clients who want to find the maximum resources available to them must choose the weakest form of allocation that their application can use.

To allocate storage at a remote IBP depot, the client calls **IBP_allocate()**. The maximum storage requirements of the byte array are noted in the **size** parameter, and additional attributes (described below) are included in the **attr** parameter. If the allocation is successful, a trio of capabilities is returned. There are several allocation attributes that the client can specify:

- **Permanent vs. time-limited** — The client can specify whether the storage is intended to live forever, or whether the server should delete it after a certain period of time.
- **Volatile vs. stable** — The client can specify whether the server may revoke the storage at any time (volatile), or whether the server must maintain the storage for the lifetime of the buffer.
- **Byte-array/Pipe/Circular-queue** — The client can specify that the storage is to be accessed as an append-only byte array, a FIFO pipe (read one end, write to another), or a circular queue where writes to one end push data off of the other end once a certain queue length has been attained.

We expect that applications making use of shared storage in a logistical network will be constrained to allocate storage that is either permanent and volatile, or time-limited and stable.

All reading and writing to IBP byte arrays is done through the four reading/writing calls in Table 1. These calls allow clients to read from and write to IBP buffers. **IBP_store()** and **IBP_load()** allow clients to write from and read to their own memory. The **IBP_copy()** call allows a client to copy an IBP buffer from one depot to another. **IBP_mcopy()** is a more complex operation. According to the **Data Mover Operation** selected (see sec. 3.2 below), it can move data to a number of end points, using different underlying protocols (TCP, UDP). The syntax of this call provides a great flexibility, allowing the research of new and non-standard ways to transfer data. Note that both **IBP_copy()** and **IBP_mcopy()** allow a client to direct an interaction between two or more other remote entities. The support that these two calls provide for third party transfers are an important part of what makes IBP different from, for example, typical distributed file systems.

The semantics of **IBP_store()**, **IBP_copy()**, and **IBP_mcopy()** are append-only. Additionally, all IBP calls allow portions of IBP buffers to be read by the client or third party. If an IBP server has removed a buffer (due to a time-limit expiration or volatility), these client calls simply fail, encoding the reason for failure in an **IBP_errno** variable.

Management of IBP byte arrays and depots is performed through the **IBP_manage()/IBP_statue()** calls. With these calls clients may manipulate reference counts, modify allocation attributes, and query the state of depots. Additionally, authenticated clients may alter the storage parameters of an IBP depot.

2.2.2 Current State of Progress in Logistical Networking Research

The initial research on Logistical Networking, including crucial development work on IBP, was funded under DOE's Next Generation Internet grant (contract # DE-FC02-99ER25396). Although three years of funding were awarded for this project, it was cut short after one year when Congress eliminated funding for the program. The current proposal encompasses the work originally proposed in the second and third years of that project, as well as work that leverages the experience already gained to go beyond the scope of the earlier proposal. Achievements of the project to date include the following:

- **Release of IBP 1.0** — Version 1.0 of the IBP client and depot has been released. The depot software, originally written by a graduate student has been almost entirely rewritten. The 1.0 API has been significantly modified from the initial API to reflect the experience of the project's first year, and it now provides much better control by the user over the dynamics of data flow through the depot. The current version is highly stable and has a fast, efficient thread-based architecture. A beta version of the IBP client library for Microsoft Windows is available, and a Windows version of the

depot is in development. A Java implementation of the client library is also planned. All IBP software is distributed as source code.

- **Test deployment of IBP Mail** — A stable version of IBP Mail (described in detail below) is currently deployed on the University of Tennessee campus and is linked to the IBP Web page (<http://icl.cs.utk.edu/ibp>). It will remain open for use as an IBP demonstration as long as storage resources hold out. The IBP Mail script is also being distributed along with simple instructions on setting up an IBP depot and IBP Mail gateway. Although IBP Mail is a very simple application, we expect it to generate substantial aggregate use of IBP depots and provide a non-trivial testbed for experimentation with logistical scheduling. It also provides an excellent testbed for the deployment of innovations such as the *external node (exNode)* implementation of network files and *Logistical Backbone (L-Bone)* based logistical scheduling (see below).
- **Registry of IBP Depots** — A registry of IBP depots is currently in prototype and will be deployed in May 2001 as the first step towards a more capable L-Bone. A Web interface will indicate the status of all registered depots, and a client library will use the information derived from this Web gateway to implement simple logistical scheduling decisions. A more capable LDAP version of this registry will be deployed by November 2001.
- **IBP-based File System** — A simple IBP-based file system that implements a directory structure and data storage completely within IBP has been developed. Its client library can also use a local IBP server for caching of data. Updates are restricted to complete replacement of a file, allowing atomic updates to be implemented through the directory. An experimental version of the Apache Web server that accesses its source files through this IBP-based file system has also been created.
- **High Performance Data Movers for Interdepot Communication** — Significant work has been done on the problem of implementing high performance Data Movers in advanced networks such as Abilene and ESNnet. The straightforward implementation of the Data Mover interface using TCP has problems scaling to networks with a high bandwidth-delay product. One solution to this problem is to make use of the results of the Web100 project, and while we are working closely with that project, results are still some time off. Other approaches include using multiple-stream TCP solutions such as GridFTP and using non-TCP protocols, but these are all problematic from the point of view of scaling and acceptance by the IETF. All approaches are being studied.
- **Network Weather Service Using IBP** — A version of the Network Weather Service (NWS) has been deployed that uses IBP depots to manage the constant flow of network measurement data from NWS sensors.
- **Logistical Session Layer Prototype Tested** — Initial results have shown LSL's strategy to be an effective mechanism to improve network performance in many circumstances. Work is ongoing in the project and a paper describing the mechanism and results has been submitted to HPDC.

3. Basic Technologies for the Storage-enabled Internet

Our work on Logistical Networking thus far has consisted of developing a mature design and a robust, efficient implementation of the IBP service, and then using the service to create simple logistical networking mechanisms within a variety of applications. The primary goals of this work have been to

- gain experience with the issues involved in taking advantage of asynchronous Internetworking, and
- obtain estimates of the performance advantages and increased functionality available through logistical scheduling.

With release 1.0, IBP has performance and stability characteristics comparable with other Internet software components (e.g., DNS, mail, news, FTP and HTTP servers, Web caches, streaming video repeaters, PKI and directory servers). While our work with applications has demonstrated both greatly improved performance and new functionality in specific instances, it has been experimental in nature and not implemented as part of the more robust and scalable software architecture that a real storage-enabled Internet would require. Much of the work proposed here, therefore, aims to build on the foundation that IBP provides to create a small,

robust set of complementary tools that can make the demonstrated benefits of Logistical Networking broadly available to the application community and scalable to the global Internet.

To understand what functionality these other components need to supply, it is important to understand how IBP could be used to demonstrate the value of Logistical Networking without requiring a robust and scalable software architecture around it. To maximize its interoperability and deployability as a low level protocol, IBP was designed to provide *minimal* abstraction of storage, with semantic properties as weak as our essential purpose would allow. This means that an IBP allocation on a particular depot exposes to the network many of the underlying characteristics of the server platform on which it runs and its associated storage resource. Thus, every IBP depot will have limits on the size and duration of allocation, and these limits will be determined by local resource bounds at the depot, as well as local policy. If a particular server platform suffers from poor network connectivity, then the byte arrays allocated by that depot will have low availability. If it suffers from frequent crashes that cause loss of stored data, then that depot's allocations will be unreliable. The design philosophy is that IBP avoids implementing features not directly supported by the underlying server platform, other than those required for integrity, security, and manageability of a network storage service.

Much of our experimentation so far has focused on the use of a single IBP depot that is assumed to possess the allocation characteristics required by the application. By controlling for or making assumptions about certain test conditions, such as server reliability and network access, we could show the benefits of using IBP without revealing its essential limitations. For example, if a very large, stable IBP depot is used in experiments, then it is not necessary to implement fragmentation across depots in order to implement large allocation, or to implement redundancy in order to improve reliability. This is analogous to using UDP in a very reliable, predictable network when TCP-like reliability and ordering properties are needed. Although it can show the value of Logistical Networking, it will not scale.

This is not to say that we have worked only in “raw” IBP mode. On the contrary, we have investigated logistical scheduling in the context of a number of specific applications, and this experience has informed our thinking about the general approaches we need to adopt going forward. An example of an application in which multiple IBP allocations have been aggregated to implement stronger properties is IBP Mail, where data is routed between an IBP depot close to the sender and depots close to the receivers [18]. Another example is the experimental IBP File system cache, which uses a reliable publication server and another IBP depot close to the reader. In each case, however, the data structures and algorithms used to implement aggregation have been built ad hoc, and the code is not easily usable in situations with different requirements.

We now propose to develop other key software elements that are required in order to create a storage-enabled Internet on a foundation of IBP. Our work will focus on four software technologies that represent new additions to the software infrastructure surround or uses IBP base, including file services built on top of IBP, interdepot transport, security and billing, and protection of IBP service providers from responsibility for stored client content (often referred to as “disintermediation”). Below we discuss each of the elements in turn.

3.1 The exNode: Aggregating IBP Storage Resources to Provide File Services

Our approach to creating a strong file abstraction on the weak model of storage offered by IBP follows a very traditional approach. In the world of end-to-end packet delivery, it has long been understood that TCP, a protocol with strong semantic properties (e.g., reliability and in-order delivery) can be layered on top of IP, a weak datagram delivery mechanism, and that the benefits of this layering have been crucial to the ubiquity of IP services. The fundamental mechanism to achieve reliability and in-order delivery of packets in spite of the weak properties of UDP is retransmission of IP packets. Retransmission, combined with protocol state maintained at the endpoints, overcomes non-delivery of packets. All non-transient conditions that interrupt the reliable, in-order flow of packets can then be reduced to non-delivery. We view retransmission as an *aggregation* of weak IP datagram delivery services to implement a stronger TCP connection.

The same principle of aggregation applies to layering a storage service with strong semantic properties on top of a weak underlying storage resource, such as an IBP depot. Strong properties such as reliability, high-performance access from a particular end-point, unbounded size and duration of allocation, will not generally be available from the underlying resource. However, with storage as with packet delivery, *aggregation* can be used to implement stronger properties:

- **Reliability** — Redundant storage of information on resources that fail independently can implement reliability (e.g. RAID, backups).
- **Fast access** — Redundant storage of information on resources in different localities can implement high performance access through proximity (e.g. caching) or through the use of multiple data paths (e.g. RAID).
- **Unbounded allocation** — Fragmentation of a large allocation across multiple storage resources can implement allocations of unbounded size (e.g. files built out of disk blocks, databases split across disks).
- **Unbounded duration** — Movement of data between resources as allocations expire can implement allocations of unbounded duration (e.g. migration of data between generations of tape archive).

In order to apply the principle of aggregation to storage services, it is necessary to maintain state that represents an aggregation of storage allocations, much as sequence numbers and timers are maintained to keep track of the state of a TCP session. In the Unix file system, the data structure used to implement aggregation of underlying disk blocks is the *inode* (*intermediate node*). Under Unix, a file is implemented as a tree of disk blocks with data blocks at the leaves. The intermediate nodes of this tree are the inodes, and they are themselves stored on disk. The Unix inode implements only aggregation of disk blocks within a single disk volume to create large files. Other strong properties are sometimes implemented in a Unix file system through aggregation at a lower level (e.g. RAID) or through modifications to the file system or additional software layers that make redundant allocations and maintain additional state (e.g. AFS, HPSS) [19, 20].

In the context of the storage-enabled Internet, we have chosen to implement a single generalized data structure for management of aggregate allocations that can be used in implementing many different strong semantic properties. This data structure, which we call an *external node*, or *exNode*, is an analogy to the inode of the Unix file system, but rather than aggregating blocks on a single disk volume, it aggregates storage allocations on the Internet. We now propose to use the exNode as the basis for building generic tools for implementing files with a range of characteristics. Because the exNode has to provide interoperability between heterogeneous nodes on a diverse Internet, we have chosen not specify it as a language-specific data structure, but as an XML serialization. The basis of the exNode is a single allocation, represented by an Internet resource: either an IBP capability or a URL. Other classes of underlying storage resources can be added for extensibility and interoperability.

The important elements to be developed are libraries that implement generic requirements such as large size (through fragmentation), fast access (through caching), and reliability (through replication). Applications requiring these characteristics should be able to obtain them even without having available individual IBP depots that implement them – simply using the APIs should be sufficient if aggregate resources are available somewhere on the network. The exNode data structure will be a basis for interoperability within the logistical networking API, and the XML serialization will be the basis of interoperability between network nodes.

Concretely, the exNode is an encoding of URLs and associated metadata in XML. The intent is for the exNode file abstraction to be used in a number of different applications. If the exNode is placed in a directory, the file it implements can be imbedded in a namespace. However, if the exNode is sent as a mail attachment, then there need not be a canonical location for it. The use of the exNode by varying applications will provide interoperability similar to being attached to the same network file system.

The exNode metadata must be capable of expressing at least the following relationships between the file it implements and constituent storage resources:

- 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 the aggregating function (simple union, parity storage scheme,)

The flexibility of the files implemented by the exNode is a function of the flexibility of the underlying storage resources. The value of IBP is not that it is the only resource that can be aggregated in an exNode, but that it is the most flexible and most easily deployed.

3.2 The Data Mover Interface

If IBP is a service that models storage as a network resource, then it is still necessary to model the movement of data between storage locations. In some sense, storage of data is more primitive than movement of data, since there are many possible data movement protocols, but each storage has a basically similar abstraction. For that reason, data movement is for us an abstraction built on top of the IBP storage model.

Typically, the movement of data into and out of a storage resource is thought of in terms of read and write operations from a producer or consumer. However, it has long been understood that network storage is different because data paths may connect storage devices directly through the network, and moving data between endpoints may be much less efficient. For this reason, network storage interfaces such as FTP, the IEEE mass storage standard [21], and IBP provide operations for direct movement of data between storage resources across the network.

The `ibp_mcopy()` call is quite general, allowing either point-to-point or point-to-multipoint communication as a single operation. Additional generality is achieved through the “operation” parameter that specifies the semantics of a particular invocation of `ibp_mcopy()`. The resulting generality is analogous to the generality of the sockets interface, which supports multiple protocols for moving data end-to-end. Because we want to allow a variety of implementations of `ibp_mcopy()` we have decided to support implementation of these operations by helper processes external to the IBP server.

The design philosophy behind our Data Mover abstraction is similar to that of IBP. A particular protocol may have very strong properties (e.g. reliability for TCP, scalability for multicast UDP). The corresponding Data Mover will display the inherent properties of the underlying protocol, but will not implement additional features. It will then be incumbent on a higher-level mechanism to aggregate Data Mover calls to implement specific properties that may fall across a wide spectrum. Thus, a reliable, scalable application may use multicast UDP along with an added retransmission mechanism, or it may use a tree built out of reliable TCP connections. In this way, with IBP the storage-enabled Internet can support a highly flexible, non-proprietary overlay network that can be used in the way that proprietary tools for distribution of live video streams are today. Other, analogous overlay functions can be built on a generic infrastructure using IBP and the Data Mover as an underlying resources.

3.3 Security and Billing

IBP was designed on the premise that security and billing should impose as little overhead as possible on the user. The model is IP networking, which implements access control only through coarse-grained network peering policies, where usage is usually unmetered and where the user is usually anonymous. This lack of fine grained access control and billing has allowed the network to grow very aggressively, to the point where wireless network is sometimes provided free and with minimal access restrictions in public places.

The IBP approach to security is twofold: The control channel between the client and depot is encrypted using SSL, to allow the capability to be passed securely. However, the management of IBP capabilities by the user is completely unrestricted, and need not involve the depot at all. The maintenance of security for capabilities is thus outside the scope of IBP. Data is passed between client and server in the clear, and if encryption is required, it must be implemented end-to-end, as with any other network infrastructure. The IBP depot makes

a best-effort attempt to keep stored data private, which may include encryption of data on the disk, but there are no strong security guarantees.

If IBP were to adopt an approach analogous to IP routing, allocation on IBP depots would be completely unrestricted. This can immediately lead to a problem with Denial Of Use attacks and misappropriation of resources. The difference between IP and IBP is that it is difficult to profitably use bandwidth in someone else's network (Peer-to-Peer content distribution is one example of a case where it is possible, but only with the donation of storage resources by users in the network). On the other hand, storage has some value no matter where it is located, as long as there is adequate connectivity.

Given that there will ultimately be a need for security, we intend to adopt a community-based approach to access lists, enabling allocation on a server according to membership in a broad community, perhaps defined by network address or perhaps by presentation of explicit credentials (certificates). Peering relationships may then be defined in a similar way, enabling communities to exchange information between their storage infrastructures. Billing would be done on the basis of community membership rather than on a per-allocation basis. A community with particularly valuable storage resources could then restrict membership and engage in limited peering, perhaps involving settlements.

3.4 Disintermediation

One important issue that has always attended the creation of a public storage infrastructure is that of responsibility for stored information. The notion of disintermediation has been well established for Internet Service Providers: if the ISP does not exert editorial control over the flow of information, then it will not be held accountable for it. This is particularly important when the information being transmitted is illegal (e.g. pirated copies of copyrighted material), immoral (e.g. pornographic) or offensive (e.g. racist). When information is stored, there is the possibility of greater impact, and there is a tendency to hold the owner of the storage resource responsible. More recently, Napster was prosecuted for simply organizing such dissemination. On the other hand, Web caches have been deemed allowable, even though they can have a negative impact on the dissemination of some Web sites. It is our hope that IBP will fall under the category of network services for which the principle of disintermediation applies. But if this is not the case, then technical and social mechanisms must be found to restrict its use to allowable applications, or to make the use of storage service traceable, aiding in the discovery and elimination of abuse.

Several modifications to the implementation of IBP are designed to strengthen the case for disintermediation. One simple step is the encryption of data on the disk. This step will make external IBP operations using the capability the only way to access stored data. If the capability directory is similarly encrypted, then there will be no simple means of examining stored data without decrypting files manually. Another simple step is to apply end-to-end encryption to the data, storing the key in the exNode data structure. Ultimately, it is impossible to publish information stored in IBP without providing a public directory (which is the function performed by Napster) and IBP does not implement any such directory. Thus, we hope that the provider of such a directory would be held responsible for any abusive publications. Without a public directory, an IBP byte array can only be shared through private communication, which does not violate intellectual property rights.

4. Research Issues and Applications

Since IBP now exists as production quality software, our research on Logistical Networking can accelerate on several extremely promising tracks. Once exNode and DataMover software have been developed as well, we believe that the positive research results we achieved will quickly translated into new functionality or better performance for SciDAC's high performance and collaborative applications. Below we describe the key lines of research that we intend to explore, along with applications that are actively driving it.

4.1 Logistical Policy for Routing and Quality of Service

The basic challenge for logistical policy can be stated as follows: *Can we develop and validate structures and algorithms to solve instances of the logistical policy problem, viz. deciding when and where to move or*

store data? In order to approach this in a systematic manner, we first describe the general problems of logistical routing and QoS in this section and then, in the subsections, discuss the various limited instances of the general problem that we intend to explore, looking for solutions to those cases that may generalize.

Traditional end-to-end networking and logistical networking share a common problem at their base — *routing*. The central enabling mechanism in IP packet delivery addresses the end-to-end routing problem: *given a packet presented to the network at point \mathbf{a} with destination \mathbf{b} , what path of intermediate nodes should it take?* Distributed routing algorithms reduce this to a local problem: *given a packet presented to an intermediate node \mathbf{a} with destination \mathbf{b} , what is the next link to which it should be passed along?* The elegance of both the end-to-end routing problem and the algorithms that allow each router's decisions to be local are at the core of the Internet's usability and scalability.

Logistical networking requires us to reformulate the end-to-end routing problem: *given an array of bytes presented to the network at point \mathbf{a} with a destination node \mathbf{b} , what sequence of data movements between \mathbf{a} and the available intermediate storage points (or data depots) should the byte array take and by which depot should a particular request for the delivery of this array be fulfilled?* We call this *the logistical routing problem*. Since this approach uses intermediate storage points, it is natural to extend logistical routing to include asynchronous logistical routing, in which the delivery time is characterized by a probability distribution $\mathbf{P}(\mathbf{t})$ over time. A different extension to simple logistical routing is multi-recipient logistical routing, in which the recipient is not simply a node but is characterized by a probability distribution $\mathbf{P}(\mathbf{b})$ over a set of potential recipient nodes \mathbf{B} . Multi-recipient logistical routing can be either unicast or multicast in nature. Finally, multi-recipient asynchronous logistical routing characterizes delivery by a joint probability distribution $\mathbf{P}(\mathbf{b},\mathbf{t})$ over a set of nodes \mathbf{B} and a range of time.

One of the complications being introduced to standard IP routing concerns the QoS of delivery of a flow of packets. QoS requirements translate to logistical routing in the following way: transfers from the sender to data depots and between depots can have QoS requirements placed on them, but the QoS of the final delivery from a data depot to the receiver is the only one which affects the receiver's perceived QoS.

One problematic point that is immediately apparent in this formulation of the logistical routing problem is the reference to a probability distribution \mathbf{P} . If treated in full generality, this distribution is a complex mathematical object that does not lend itself to efficient analysis and decision-making [22, 23]. In many situations little will be known about the true distribution. For these reasons, a highly simplified space of distributions must be used in practical situations, and it may contain little information.

To make logistical routing decisions it is necessary to have global information about resource availability and a model of future demands for particular data, although this information can be approximate and/or aggregate in nature. Making information available about global resource availability is exactly the function of the Network Weather Service (NWS) [10, 24, 25], which is a core component of our work already in place. There are many ways to model future demand, from explicit reservation of requests, to the use of dynamic mechanisms based on locality of reference such as caches, to application-specific models which operate at run time but anticipate application demands wherever possible, as in AppLeS [26]. Part of our research will be examining the usefulness of various logistical scheduling models. *The goal of this portion of our research is to find a distributed algorithm for QoS-enabled logistical routing which uses only information that has been localized at each depot to choose the next depot in the path of transfers.*

4.2 Buffering Data at Network Boundaries: The Logistical Session Layer

When sending data directly from a sender to a receiver, it is sometimes possible to identify locations where, due to mismatches in transmission characteristics across network boundaries, it would be advantageous to increase the buffering of the data in FIFO fashion. This allows the storage server to act as *both* surrogate sender and surrogate receiver, creating new optimizations of typical uses of networking.

We call one version of this strategy that we intend to explore the *Logistical Session Layer (LSL)*. LSL is an application of logistical networking designed to address problems of bulk data transfer over networks with high bandwidth/delay products. Despite the many advances in TCP that have been made to address this

situation, there is still a fundamental cost associated with buffering unacknowledged segments for retransmission. Moreover, it is clear that the problem is only exacerbated as network speeds increase. For instance, sufficient buffering to support a TCP stream that half fills an OC-48 link from the east to west coast must be on the order of 15 megabytes, likely much higher. Following an approach which is similar to that proposed by Salehi *et al* for use in multicast video transmission [27], LSL will make use of IBP depots to ensure that a packet loss need not require a retransmit from the original source, but rather may do so from an intermediate location.

LSL will be a “session” layer (layer 5) in terms of the OSI protocol model. A connection that is initiated through the LSL can pass through a number of IBP depots. In this scenario these depots can actually be thought of as “transport layer routers.” Recall that a transport layer conversation consists of multiple hops of network layer conversations. In an analogous fashion, we envision a session layer conversation to consist of multiple hops of IBP data transfers. This session-oriented approach has the dual benefits of utilizing the established protocols at lower layers and of encapsulating many of the administrative details that have to be managed when working with IBP directly.

The LSL interface will closely mimic the Berkeley socket interface. In this way, existing programs that might benefit from this intermediate buffering can be easily modified to take advantage of it without having to use the IBP API directly. The familiar socket calls can be modified to implicitly utilize IBP buffers like routing nodes along the path. These logistical sockets will initially be half duplex, and both the source and destination port of such a half-duplex connection will be assigned by the initiator. This will necessitate that the `lsl_bind()` on the receive side be performed without specifying a local port.

When an LSL connection is initiated, a predicted path may be specified or local forwarding decisions may be relied upon. To specify a path explicitly, the sender will use the strict source route options with the LSL socket. In fact, a combination of local and global forwarding strategies may be employed by specifying a loose source route in the same fashion. *The goal of this portion of our research is to implement an LSL interface and setup protocol that will significantly improve the performance of data transfer on paths that cross network boundaries with significantly mismatched characteristics.*

4.3 Parallel I/O for distributed applications

The MPI_connect package [28] allows multiple distributed HPC MPI applications to be joined/merged at the message passing level to create larger meta-applications. One of the systems this package is used for is the DOD MSRC MetaQueuing facility, which allows a single point to submit a job that could be executed at a number of distributed sites. One feature of the MetaQueuing system was the user was not aware of where the parallel application would execute until runtime. This had the side effect that any large input files required would have to be replicated at multiple sites in advance. To avoid this MPI_Conn_IO was developed to allow for on-demand downloading of file data by intercepting open calls within the MPI Parallel IO API retrieving the necessary files. This avoided the need to replicate files or to use a complex distributed file system such as AFS, while still allowing applications to access global globally distributed data.

An important feature of this system was its ability to move very large files in the most efficient method via a number of possible paths to avoid network bottlenecks. Figure 3 above shows an example application executing at site A (ERDC), with the application data stored at site B (ORNL). A third site C (UT) provided a caching service based on IBP, and thus an alternative networking route. The network performance between

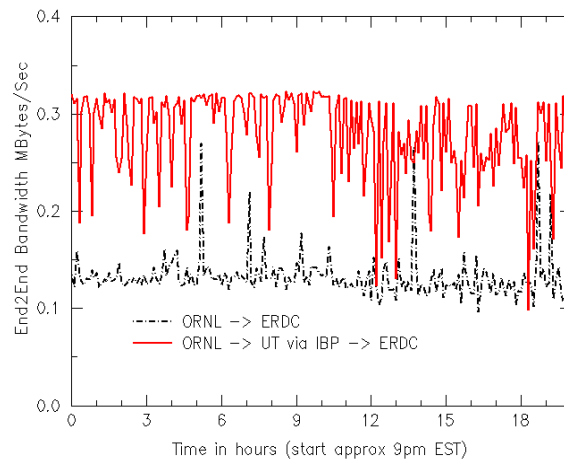


Figure 3: IBP for Parllel I/O

A-B is poor compared to the connectivity of the other paths (A-C, B-C). Thus MPI_Conn_IO would automatically use the IBP caching service to allow it to route the large input file to the application quicker than the directly routed path taken by an end-to-end TCP connection. The graph above shows how the end-to-end network bandwidth utilized by moving a large file between these different routes varies during a twenty hour period with a sample being taken ten times an hour. As can be seen the use of IBP improves performance by over a factor of two when transferring large files.

4.4 IBP-Mail

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. We have already implemented a proof of concept prototype of this application [18] and now have a stable version in test-bed deployment on our campus. Our research will exploit recipient addresses and forwarding patterns to infer the delivery information required for logistical routing.

IBP-Mail allows a mail attachment to be passed between users in the following manner: user A executes code

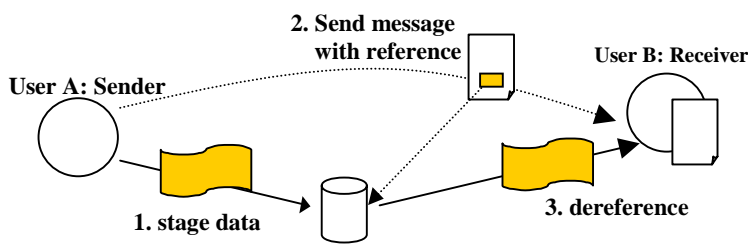


Figure 4: IBP Mail

that finds a suitable IBP server in which to store the file, stores the file there, and then sends the capabilities (soon to be exNodes) to user B in a MIME attachment (Figure 3). Upon receiving the attachment, user B's mailer launches a program that downloads the file from the IBP server. File deallocation at the IBP server may be performed either via the time-limited allocation feature, or by sending user B the

management capability, and having user B deallocate the file. We have also implemented a very simple form of file routing in IBP-Mail: user A inserts the file into an IBP buffer on a depot close to his system, and then the buffer is moved asynchronously to an IBP buffer close to user B. This gives fast insertion for user A and fast delivery to user B.

Once we have integrated the much more flexible exNode technology with this basic functionality, our experimentation with IBP-Mail will focus on the following issues:

- **Forwarding** — *Can we extend the IBP-Mail routing paradigm to enable intelligent decisions about whether attachments should be copied when forwarding?* Passing exNodes instead of copying implements an asynchronous form of multicast.
- **Processing in the network** — Another extension would augment the handling of IBP attachments to allow them to be processed (e.g. by compression/decompression) without requiring them to be downloaded to an end-user system. *Can such a scheme be used to create a flexible system for management and processing of data in the network, enabling workstations to direct the handling of data sets they cannot even store?*

4.5 Resolution for Highly Volatile Resources

The challenge of providing a general resolution mechanism for logistical networking can be stated as follows: *Can we develop and validate structures and algorithms to construct a reliable, high-performance system for discovery and access to replicated and distributed data resources that are individually unreliable or of potentially low performance?*

Data stored in an exNode may vary in the strength of semantic guarantees associated with it, depending on allocation attributes:

- An exNode that implements a highly stable file can be stored in a persistent directory, serving the function of a Unix file system inode.

- A shorter-lived exNode can be associated with processes that have a limited duration (such as the transfer of e-mail).
- Highly volatile resources create a special problem, since bindings between names and capabilities or even sets of capabilities can quickly become unusable due to “broken links.” Redundancy can be used to obtain an expected performance and reliability profile from a set of capabilities that individually would otherwise be unreliable or have poor performance. However, if the stored data changes, this creates a problem of maintaining consistency between unreliable and low performance storage resources.

Traditional directory services, such as file systems, DNS [29] and LDAP [30], which bind long-lived names to sets of capabilities, work well with resources that are stable and reliable. The IETF URN project [31] is one example that sought to create robust names by managing a set of less robust capabilities, but to date such projects have not been successful.

We propose to investigate an approach to manage sets of capabilities that may have very weak semantics without creating long-lived names that are bound to sets of capabilities. This can be done by indexing the capabilities according to a set of attributes and access them by selecting from a large available space. The strength of such an approach is that the set of capabilities can change, and can be different for different users, but correctness depends only on the semantics of the attributes. The weakness of this approach is that the correctness depends on the correctness of the attributes. This approach has been used in distributed systems such as Linda [32, 33] and in mass storage systems such as the Storage Resource Broker [34].

Napster is an example of a system that aggregates very weak capabilities to create a reliable service (<http://opennap.sourceforge.net/napster.txt>). Napster is a system that allows Internet users to cooperate to create a digital library of MP3 music files. The files are stored on end-user workstations, and so access to them may be very unstable. Each workstation registers with a server, and queries are directed to the server that match against such attributes as artist and track name. A list of available queries is returned, and this list may be different for every query. As long as the attributes are correct and the search criteria are specific enough, this scheme can create a very robust service for accessing data. If such integrity criteria are not enforced, however, Napster can be expected to experience the same loss of focus in the results of searches currently being faced by Web search engines.

The goal of our work on this front will be to create a system that has the predictability of Unidata [35] or SRB [36] while modeling a much broader class of possible attributes for the selection of data. We hope to approach the open structure of systems like Napster without running into the problem of diminishing focus of searches.

4.6 Application-specific and Generalized Cache Management

IBP is a primitive abstraction of storage, so it implements no transparency of location. There is a large class of applications for which the semantics of IBP capabilities are sufficient, except for the lack of location transparency, which means that it does not natively support caching to take advantage of locality. However, we have already used it to implement application-specific caching for NetSolve and the results have been impressive.

NetSolve [11] is a software environment for networked computing that is currently in use at many institutions. The design of NetSolve allows users to make use, via the network, of a variety of computational resources at their disposal using familiar interfaces from the world of uniprocessor computing (e.g. Matlab, simple procedure calls). NetSolve uses a *client-agent-server* paradigm. NetSolve users are the *clients*, and the computational resources are the *servers*. A server may be a uniprocessor, a MPP (Massively Parallel Processor), or a networked cluster of machines. When a user wants a certain computational task to be performed, he/she contacts an *agent* with the request. Each agent maintains information such as availability, load, and supported software, on a collection of servers. When a request from a user comes in, the agent selects (on the basis of server load, available bandwidth, etc.) a server to perform the task, and the server responds to the client's request.

The user's computation is performed remotely at the server. This means that the user sends his/her data to the server, and the server uses its own software to perform the computation. When the server completes the computation, it sends the result back to the client, and contacts the agent to notify it of the completion of the task.

NetSolve's basic computation model is functional: servers receive all of their input from the clients and then return all the results to the client. There is currently no management of global state in NetSolve and this is a limitation that can significantly constrain its performance. However, a recent extension of the NetSolve computational model uses IBP to implement caching of data at the server. This caching enables large data objects so that repeated uses can avoid paying the cost of repeatedly fetching them across the Internet. Initial experiments show that the speedup from this optimization is linear in the size of the cached data object, with a slope of $2/3$ for objects of size 1-3MB (speedup of 1-3). Extrapolating this performance to larger objects, a 10MB object would see a speedup of 6, and a 100MB object would see a speedup of over 60.

We will implement a generalized cache management service that will support a parameterized class of storage management policies, e.g. write-back vs. write-through, dynamic vs. preloaded, etc. These caches will support hierarchical and cooperative arrangements to create scalable storage management systems. This will provide NetSolve and many other grid-enabled applications with direct and facile access to application independent caching capabilities. Using IBP the storage-enabled Internet will be able to make application independent support for caching, with all its attendant performance enhancements, a routine matter.

Many of these applications will use IBP buffers with a lifetime of hours or less. The sources of the instability may vary, from time-dependent data that simply is not worth storing for longer periods to servers located at network locations with unstable connectivity. In these cases the exNode may be used as a standard format for the transfer of volatile IBP capabilities implementing a stored resource, but this information may be transactional and not be worth storing for any period of time.

Examples of applications which can make use of such a lightly cooked IBP include the following: meteorological data collection systems such as Unidata [35, 37]; network monitoring and analysis systems such as the Network Weather Service [10] which monitors network resources and predicts their future behavior; highly distributed file distribution systems such as Napster (<http://opennap.sourceforge.net/napster.txt>); and Web-like electronic publications.

4.7 Architecture

The efficiency of modern implementations of IP routing derives from the fact that they are embedded in operating system kernels and special purpose routers. It is possible that naïve implementations of logistical networking will compare badly with typical IP routing due simply to inefficient software architecture. So the question is *can we implement our logistical networking mechanism in a manner which can operate at high performance without disrupting the function of the network by introducing instability, unduly complicating network management or imposing excessive resource overhead?*

In Logistical Networking we decompose interactions currently taking place over a single TCP/IP connection into the *conversational* component, which travels over TCP/IP, and the *contextual* component, which is delivered using IBP (see sec. 2.1). In so doing we introduce a new communication service that is not a part of the TCP/IP protocol stack and is not implemented by current operating system kernels. The most straightforward implementation of this new service requires the use of a server process that is external to the application and which is itself accessed using TCP/IP connections. While this gives us great flexibility in the placement of the IBP depot, it introduces an additional copy of information between processes, which introduces overhead that can adversely affect application performance.

An obvious response is not to implement IBP in this naïve way, but to take one of many possible optimized approaches. Part of our research is to examine the performance of the current implementation of IBP in various applications to determine which optimizations are appropriate.

- One approach is to use an operating system specific mechanism for the sharing of memory between processes. While this has the disadvantage of making the implementation of the IBP depot operating system specific, it can eliminate the extra inter-process copy.
- Another approach is to situate an IBP depot within the client process by allowing the IBP client library to function as a server. This approach requires a thread-based operating system in order to allow IBP requests to be interleaved with application processing, and that also is not a standardized operating system feature.

4.8 Depot Policies and Allocation Strategies

In IBP, the willingness of a depot to make a requested allocation is determined by a local policy. Many factors come into play, including the current and anticipated future demands on the storage resources available to the depot, the size, time limitation, and stability/volatility of the requested allocation. We anticipate the development of a policy that is increasingly conservative in allowing allocations as storage resources approach capacity.

The application has two ways of sensing the state of storage resources in the network: implicitly through rejection of allocation requests by IBP depots, and explicitly through information disseminated through the `IBP_status()` call. The application can respond in one of two ways: by backing off in its use of storage, or by fragmenting its request and requesting resources from a larger pool of depots, perhaps making greater use of shorter duration or volatile resources. The former approach is only possible in applications that can throttle their use of storage resources. The latter approach leads to the greater possibility of a faults or poor performance in a depot holding a fragment or in the network connecting that depot.

Given a model of application requests, application strategies and depot policies, we can ask whether the IBP network storage fabric will be stably usable, particularly under conditions of high load or congestion. While it is outside of the expertise of the PIs to address this question analytically, simulation and experimental approaches are feasible. We intend to devise a number of policies and to test them in a simulated environment, using standard tools for discrete event simulation.

5. The Logistical Backbone (L-Bone): Deployment of Logistical Networking on the I2-DSI

IBP models the fundamental structure of the Internet at the level of network locality. In order to be of maximum use, it must be deployed across a variety of localities, allowing it to be used for management of stored data and computational state among those localities. We are following the usual deployment strategy, which is to make open source software freely available to the Internet community. We are also following a second strategy: we are establishing IBP depots on the servers being deployed by the *Internet2 Distributed Storage Infrastructure (I2-DSI)* project [39], creating the first nodes of an experimental testbed for logistical network that we call the *Logistical Backbone (L-Bone)*. Since I2-DSI will supply the initial foundation for the L-Bone, its nature and status need to be understood first.

The I2-DSI project is another LoCI project that is experimenting with the logistical use of storage and locality in networking, but one which focuses on services delivered at the application level, such as Web protocols, video streaming, and various kinds of data set manipulation. As part of the investigation of logistical networking, the deployment of I2-DSI requires a broadly distributed infrastructure consisting of large storage servers. We will leverage the Innovative Computing Laboratory's (ICL) leadership of the I2-DSI project to make use of these servers to deploy IBP depots that will be freely available for use by the Internet2 community. This aggressive deployment strategy will put IBP services into people's hands as soon as they obtain the client software, much as the early Web was available to anyone with a Web browser, except the resources served up by IBP are writable and can be used in flexible and powerful ways.

I2-DSI is a joint project of the University of Tennessee's Innovative Computing Laboratory, the University of North Carolina's School of Information and Library Science and the University Corporation for Advanced Internet Development. PI Beck is the I2-DSI Project Director. The project has received funding from the NSF under contract # ANI9980203, North Carolina Networking Initiative, the Digital Library Federation and seven industrial sponsors: Cisco Systems, Ellemtel, IBM, Novell, StorageTek, Starburst Communications,

and Sun Microsystems. The project is in the early stages of deployment, with 5 servers donated by IBM installed and equipped with terabyte storage systems and one server donated by StorageTek with 700GB of disk space. Software development personnel donated by Ellemtel have led to a spin-off company based on technology developed within the I2-DSI project, Lokomo Systems (<http://www.lokomo.com>). More details can be found at <http://dsi.internet2.edu>. Interest in the I2-DSI project is high in the international networking community, in particular the TransEuropean Research Networking Association (TERENA), and the Asia Pacific Advanced Network (APAN).

The combination of the technology resources (current and near future) deployed on I2-DSI and the research and development community has created an excellent foundation for building the logistical networking testbed, the *L-Bone*, that our proposed research requires. In general technical terms, the LBone consists of

- IBP depots and auxiliary utilities installed on a set of nodes including the I2-DSI core nodes managed by the ICL, and
- additional IBP depots installed and managed by our research partners in the LBone project.

The logistical networking capabilities supported on this infrastructure will include NWS sensing of the storage and network resources, IBP caches, and state management for IBP Mail inclusions. But just as important for the progress of our research is the growing application community for I2-DSI (see <http://dsi.internet2.edu/apps99.html>), which we believe will provide an excellent nucleus of users for the logistical networking applications we want to experiment with on the L-Bone.

6. Related Work

A number of data intensive computing projects, including the San Diego Storage Resource Broker [34] and the Distributed Parallel Storage System at Lawrence Berkeley Nation Laboratory [40] implement coarse-grained data staging, usually synchronous and explicitly scheduled by the application. These systems improve the performance of programs whose computation patterns and use of data benefit from coarse-grained communication profitable.

A number of companies such as i-drive (<http://www.idrive.com>) have emerged offering free access to hosted storage over the Internet. These companies offer storage at a few data centers, and are thus similar to IBP in that they make some storage freely available, but very different in that they cannot make use of locality in allocation. Also, their business plans involve offering proprietary services for data stored within their storage pool, and they do not offer interoperability between storage pools. Thus, they represent isolated resources that are balkanized in order to obtain commercial advantage. In contrast, IBP allows an ISP a means to provision their network to serve their users and interoperate with other storage-enabled networks.

Also in the past few years, commercial Storage Area Networking [SAN] and Network Attached Storage [41] have gained attention within the networking community. These system architectures are based on the concept of extending private interfaces across Local Area Networks; either the SCSI disk interface in the case of a FiberChannel SANs or the NSF remote mount interface in the case of Network Attached Storage. Efforts to extend these interfaces or build similar SAN interfaces that can be extended across wide area IP, such as the IETF IP Storage Working Group [42] networks, focus on private resources within enterprise networks rather than publicly shared resources.

The premise of *active networking* is that network intermediate nodes will increasingly participate in computation. The LoCI view, similar to that of Gibson's Active Disks/Networks [43] work, is that the role of the active network should be to schedule high performance work at the periphery, increasing the predictability of wide area computation.

Metacomputing systems are those that tie together large numbers of heterogeneous processing elements, perhaps across a wide area, in order to achieve high performance. Examples are Globus [12] and Legion [44]. Every metacomputing system implements storage management as part of its runtime system. Part of the Globus system is a storage management system called GASS [45], which manages movement of input and output files from system to system, as well as pre-fetching of input files and write-back of output files. Similarly, Legion has Shared Persistent Spaces [46]. But IBP is different from such methodologies in that it is primitive middleware, and so can span application environments and can be highly predictable in its operation.

7. Plan of Work

Like TCP/IP itself, the success of IBP and logistical networking can only be evaluated in terms of the middleware and, ultimately, the end-user applications that are enabled by them. While a clear understanding of the engineering characteristics of these tools is necessary, the tools must fundamentally model network resources in a way that can be used by application communities.

For this reason, our research approach is to develop the basic logistical network engineering tools, primarily IBP and the exNode, along with middleware, infrastructure and a few prototype applications. We seek to test our hypothesis concerning the expressive power of logistical networking and the importance of state management in distributed computation and information systems, not simply develop another sterile demonstration for the laboratory or closed testbed. In fact we believe that, because logistical networking and asynchronous communication is so fundamental to robust and scalable data sharing, the technology we are developing will quickly show itself to be a crucial part of the Collaborative Software Infrastructure on which the success of SciDAC depends. Therefore we want to move as aggressively as possible to get logistical middleware out of the laboratory and into the hands of SciDAC users.

As we detail in the table below, in the first 6 months, most of our work will focus on developing robust, scalable logistical networking middleware and deploying it across the L-Bone, which is a testbed consisting of our core I2-DSI machines and those of a few key research partners (see section 6). Throughout the first 18 months of our work we will be building the prototype applications, working with partners in the research community to validate the importance of logistical networking in the overall structure of distributed systems, and moving them to using our second generation, exNode based logistical networking middleware. In the last 18 months of the project we will focus on developing the policies required for a large-scale deployment of our technology on advanced networks like ESNNet, as well as advanced architectural models to offer performance and control of resources that are not available from our early software products.

The use of an actual testbed deployment in a limited community must be emphasized here. This project is not of a scale to offer a community-wide deployment, but it seeks to lead the way in deploying services that the community requires, so as to stimulate the development of a full-scale, second generation testbed across the NGI networks. The history of multicast and the MBone shows us that performance of individual systems or isolated testbeds does not predict the experience of a diverse community. Our ability to leverage the I2-DSI/L-Bone testbed for our research is one of the benefits of the academic community being organized for the purpose of advanced technological development.

TIME FRAME	SOFTWARE DEVELOPMENT MILESTONES
3 mos.	Continuing development of IBP depot; deployment of early L-Bone tools Initial development of exNode libraries; IBP Mail available to users
6 mos.	exNode based IBP Mail and other IBP applications demonstrated at SC'01 exNode support in NetSolve
9 mos.	Reliability/performance coscheduling alpha Simulation of allocation policy
12 mos.	Initial deployment of generalized caching infrastructure Initial deployment of LSL-based logistical overlay network on ESNNet
18 mos.	Development of wide-area logistical peering mechanisms and policies Resolution for highly volatile storage resources
18-36 mos.	Experimental IBP architectures Large scale measurement and simulations

Bibliography

1. *Scientific Discovery through Advanced Scientific Computing*. 2000, Office of Science, U. S. Department of Energy: Washington, D.C. p. 24.
2. Langer, J., et al., *National Workshop on Advanced Scientific Computing*. 1998, U. S. Department of Energy and the National Science Foundation: Washington, D. C. p. 22.
3. Kennedy, K. and B. Joy, *Information Technology Research: Investing in Our Future*. 1999, President's Information Technology Advisory Committee: Washington, DC. p. 1-80.
4. Saltzer, J.H., D.P. Reed, and D.D. Clark, *End-to-End Arguments in System Design*. ACM Transactions on Computer Systems, 1984. **2**(4): p. 277-288.
5. Plank, J., et al. *The Internet Backplane Protocol: Storage in the Network*. in *NetStore99: The Network Storage Symposium*. 1999. Seattle, WA.
6. Beck, M., T. Moore, and J. Plank, *Logistical Networking: Sharing More Than the Wires*, in *Active Middleware Services*, S. Hariri, C. Lee, and C. Raghavendra, Editors. 2000, Kluwer Academic Publishers: Boston. p. 240.
7. Zhang, L., et al., *RSVP: A New Resource ReSerVation Protocol*. IEEE Network, 1993. **7**(5).
8. Clark, D., S. Shenker, and L. Zhang. *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism*. in *SigComm'92*. 1992. Baltimore, MD.
9. Blake, S., et al., *An Architecture for Differentiated Services*. 1998, IETF.
10. Wolski, R., *Forecasting network performance to support dynamic scheduling using the Network Weather Service*, in *Proceedings of the 6th IEEE Symposium on High Performance Distributed Computing*. 1997, IEEE Computer Society Press: Los Alamitos, CA. p. 316-325.
11. Casanova, H. and J. Dongarra, *Applying NetSolve's Network Enabled Server*. IEEE Computational Science & Engineering, 1998. **5**(3): p. 57-66.
12. Foster, I. and K. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*. International Journal of Supercomputer Applications, 1997. **11**(2): p. 115-128.
13. Beck, M., et al., *Logistical Quality of Service in NetSolve*. Computers and Communications, 1999. **22**: p. 1034-1044.
14. Cerf, V., et al., *National Collaboratories: Applying Information Technology for Scientific Research*. Computer Science and Telecommunications Board (CSTB). 1993, Washington, D. C.: National Academic Press.
15. Crowcroft, J., et al., *A Rough Comparison of the IETF and ATM Service Models*. IEEE Networks, 1995. **9**(6).
16. Banerjee, A., et al., *The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences*. 1994, International Computer Science Institute: Berkeley, CA,.
17. Bassi, A., et al., *Internet Backplane Protocol: API 1.0*. 2001, Department of Computer Science, University of Tennessee: Knoxville, TN.
18. Elwasif, W., et al. *IBP-Mail: Controlled Delivery of Large Mail Files*. in *NetStore99: The Network Storage Symposium*. 1999. Seattle, WA.
19. Morris, J.H. and e. al., *Andrew: A Distributed Personal Computing Environment*. Communications of the ACM, 1986. **20**(3): p. 184-201.
20. Watson, R.W. and R.A. Coyne. *The Parallel I/O Architecture of the High-Performance Storage System (HPSS)*. in *IEEE Mass Storage Systems Symposium*. 1995: IEEE Computer Society Press.
21. Garrison, R., *Reference Model for Open Storage Systems Interconnection: Mass Storage Reference Model Version 5*. 1994, IEEE Storage System Standards Working Group (P1244).

22. Harchol-Balter, M. and A. Downey, *Exploiting Process Lifetime Distributions for Dynamic Load Balancing*, in *Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*. 1996, ACM.
23. Wolski, R., N. Spring, and J. Hayes. *Predicting the CPU Availability of Time-shared Unix Systems on the Computational Grid*. in *HPDC8*. 1999.
24. Wolski, R., *Dynamically Forecasting Network Performance Using the Network Weather Service*. Cluster Computing, 1998.
25. Wolski, R., N. Spring, and J. Hayes, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*. Future Generation Computer Systems (to appear), 1999.
26. Berman, F., et al., *Application-level scheduling on distributed heterogeneous multiprocessor systems*, in *Proceedings of Supercomputing '96*. 1996.
27. Salehi, J.D., et al. *Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing*. in *ACM SIGMETRICS*. 1996. Philadelphia, PA.
28. Fagg, G.E., K.S. London, and Jack J Dongarra, *MPI_Connect, Managing Heterogeneous MPI Application Interoperation and Process Control*, in *Lecture Notes in Computer Science*. 1998., Springer Verlag. p. 93-96.
29. Mockapetris, P., *Domain Names -- Concepts and Facilities*. 1987, IETF.
30. Howes, T. and M. Smith, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*. 1997: Macmillan Technical Publishing.
31. Sollins, K., *Architectural Principles of Uniform Resource Name Resolution*. 1998, IETF.
32. Carriero, N. and D. Gelernter, *Linda in Context*. Communications of the ACM, 1989. **32**(4): p. 444-459.
33. Carriero, N. and D. Gelernter, *How to Write Parallel Programs*. 1990, Cambridge, MA: MIT Press.
34. Group, S.S., *SRB - The Storage Resource Broker Manual (Version 1.1)*. 1998.
35. Bates, S. and D. Fulker. *Unidata Internet Data Distribution (IDD)*. in *Proceedings, Tenth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*. 1994. Nashville, Tennessee: American Meteorological Society.
36. Baru, C., et al. *The SDSC Storage Resource Broker*. in *CASCON'98*. 1998. Toronto, Canada.
37. Cooper, C.S., *Design of a Communications Network Subsystem for the Unidata Project*. Journal of the Institution of Electronic and Radio Engineers, 1987. **57**(3): p. 96-100.
38. IBM, *Replication of Nagano Olympic WWW Sites*. 1998.
39. Beck, M. and T. Moore. *The Internet2 Distributed Storage Infrastructure Project: An Architecture for Internet Content Channels*. in *3rd International WWW Caching Workshop*. 1998. Manchester, England.
40. Johnston, W. *High-Speed, Wide Area, Data Intensive Computing: A Ten Year Retrospective*. in *7th IEEE Symposium on High Performance Distributed Computing*. 1998. Chicago.
41. Gibson, G. and R.V. Meter, *Network Attached Storage Architecture*. Communications of the ACM, 2000. **43**(11): p. 37-45.
42. *Charter: IP Storage (IPS) Working Group*. 2001, Internet Engineering Task Force.
43. Riedel, E., G. Gibson, and C. Faloutsos. *Active Storage For Large-Scale Data Mining and Multimedia*. in *24th International Conference on Very Large Databases (VLDB '98)*. 1998. New York, NY.
44. Grimshaw, A., W. Wulf, and e. al., *The Legion vision of a worldwide virtual computer*. Communications of the ACM, 1997. **40**(1): p. 39-45.

45. Bester, J., et al. *GASS: A Data Movement and Access Service for Wide Area Computing Systems*. in *Sixth Workshop on I/O in Parallel and Distributed Systems*. 1999.
46. Lewis, M.J. and A. Grimshaw, *The core legion object model*. Fifth International Symposium on High Performance Distributed Computing, 1996.