The Logistical Backbone: Scalable Infrastructure for Global Data Grids

Alessandro Bassi†, Micah Beck†, Terry Moore†, James S. Plank †
†LoCI Laboratory - University of Tennessee
203 Claxton Building - 37996-3450 Knoxville, TN, USA
[abassi, mbeck, tmoore, plank]@cs.utk.edu

Abstract

Logistical Networking [1] ***can be defined as the global optimisation and scheduling of data storage, data movement, and computation. It is a technology for shared network storage that allows an easy scaling in terms of the size of the user community, the aggregate quantity of storage that can be allocated, and the distribution breadth of service nodes across network borders.***

After describing the base concepts of Logistical Networking, we will introduce the Internet Backplane Protocol, a middleware for managing and using remote storage through allocation of primitive "byte arrays", showing a semantic in between buffer block and common files. As this characteristic can be too limiting for a large number of applications, we developed the exNode, that can be defined, in two words, as an inode for the for network distributed files. We will introduce then the Logistical Backbone, or L-Bone, is a distributed set of facilities that aim to provide high-performance, location- and application-independent access to storage for network and Grid applications of all kind.

Keywords: Logistical Networking, IBP, storage-enable Internet

I. Introduction

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. An indispensable key to realizing this vision, though, 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.

At the base of the Logistical Networking project [10] 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 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 represents a more radical approach, trying to get maximum leverage out of the ongoing, exponential growth in all types of computing resources - processing power, communication bandwidth, and storage. Logistical Networking can thus be defined as 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, it is possible for application designers, and in particular grid appliction 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.)

A common reaction to the storage-enabled Internet idea is that many of these functions could be served by using individually owned resources accessed by various protocols already established. Examples are mail

[1] This work is supported by the National Science Foundation under Grants ACI- 9876895, EIA-9975015, EIA-9972889, ANI-9980203, the Department of Energy under the SciDAC/ASCR program, and the University of Tennessee Center for Information Technology Research

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.

This paper is organized as follows: section II descroibes the the Internet Backplane Protocol, which can be considered the basic mechanism for any work in the Logistical Networkinging area. Section III presents a file abstraction for storage in the wide area, called the exNode, a necessary brick to build in the robustness, ease of use, and scalability that the storage-enabled Internet will require. Section IV describes the Logistical Backbone, while Section V presents some applications and projects that already use the Logistical Networking conceps and the IBP software in particular.

II. The Inernet Backplane Protocol

IBP [8] is middleware for managing and using remote storage. 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 allows 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 Lo-

gistical 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.

A. 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 overcommitting 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 cryptographically secure URLs, called *capabilities* to the client: one for reading, one for writing, and one for management. 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 [7] and is available at http://loci.cs.utk.edu/ibp/documents.

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

TABLE I
IBP API CALLS

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 are included in the attr parameter. If the allocation is successful, a trio of capabilities is returned.

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, which utilises a Data Mover plug-in module to 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() and 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.

B. IBP Implementation

1) Depot: The main IBP depot architecture goals were identified as flexibility, reliability and performance. The software architecture of the current IBP depot implementation (1.2) is a multi-threaded one, with a pool of threads created at boot time, in order to have good performance results. The code base is shared between Unix/Linux/OS X and Win32 versions, and between file system and RAM based depot. The I/O calls are encapsulated, and two different libraries (for the File System and for pinned RAM memory) have been created; this strategy allows us to concentrate on the general design of the implementation, and to have always versions on sync.

2) Client Library: The IBP Client Library, offered in a few different versions and systems, was designed to be flexible, to ease the implementation of future changes to both the API and the protocol, to be very maintainable code and to be extremely robust and fault-tolerant. In order to satisfy these three goals we decided to separate our client library into two different modules: the API2P Module and the Communication Module.

The first module translates the API command into Communication Units, which are abstract data types that specify the communication and its characteristics (direction, semantics of the message, the message itself or the expected message). Then, the ComModule allows the execution of the communication. No analysis of the message is made at this level, the API2P module being responsible to interpret the message and to take the appropriate action. This design allows easy changes to the API (as it's seen as a sequence of communication units) and to the protocol.

3) Protocol: The version 1.0 of the protocol is currently on a final draft phase. We intend to publish the specification in the comiong months within organisations such as the Global Grid Forum [3] or IETF [4]. The protocol had to be designed from scratch as there was no current framework that allowed, explicitely or implicitely, the allocation of space at remote network appliances.

III. 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 continues to parallel the design paradigm of the traditional network stack. In the world of end-to-end packet delivery, it has long been understood that TCP, a protocol with strong semantic properties, such as reliability and in-order delivery, can be layered on top of IP, a weak datagram delivery mechanism. Retransmission controlled by a higher layer protocol, 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 can be applied in order to layer a storage service with strong semantic properties on top of a weak underlying storage resource that does not generally provide them, such as an IBP depot. Examples of aggregating

weaker storage services in order to implement stronger ones include the following:

- **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 [17]).
- **Unbounded allocation:** Fragmentation of a large allocation across multiple storage resources can implement allocations of unbounded size (e.g. files built out of distributed 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 this exposed-resource paradigm, implementing a file abstraction with strong properties involves creating a construct at a higher layer that aggregates more primitive IBP byte-arrays below it. To apply the principle of aggregation to exposed storage services, however, it is necessary to maintain state that represents such an aggregation of storage allocations, just as sequence numbers and timers are maintained to keep track of the state of a TCP session. Fortunately we have a traditional, well-understood model to follow in representing the state of aggregate storage allocations. 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, which are themselves stored on disk. The Unix inode implements only the aggregation of disk blocks within a single disk volume to create large files; other strong properties are sometimes implemented 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 [16], HPSS [15])

Following the example of the inode, we have chosen to implement a single generalized data structure, which we call an external node, or exNode, in order to manage of aggregate allocations that can be used in implementing network storage with many different strong semantic properties. Rather than aggregating blocks on a single disk volume, the exNode aggregates storage allocations on the Internet, and the exposed nature of IBP makes IBP byte-arrays exceptionally well adapted to such aggregations. In the present context the key point about the design of the exNode is that it has allowed us to create an abstraction of a network file to layer over IBP-based storage in a way that is completely consistent with the exposed resource approach. The exNode is the basis for a set of generic tools for implementing files with a range of characteristics. Because the exNode must provide interoperability between heterogeneous nodes on a diverse Internet, we have chosen not to specify it as a language-specific data structure, but as an abstract data type with an XML serialization. The basis of the exNode is a single allocation, represented by an Internet resource, which initially will be either an IBP capability or a URL. Other classes of underlying storage resources can be added for extensibility and interoperability.

Despite our emphasis on using an exposed-resource approach, it is natural to have the exNode support access to storage resources via URLs, both for the sake of backward compatibility and because the Internet is so prodigiously supplied with it. It is important to note, however, that the flexibility of a file implemented by the exNode is a function of the flexibility of the underlying storage resources. The value of IBP does not consist in the fact that it is the only storage resource that can be aggregated in an exNode, but rather that it is by far the most flexible and most easily deployed.

IV. The Logistical Backbone: Deployment of Logistical Networking

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 in a number of institutions, with particular regards to the Internet2 Distributed Storage Infrastructure (I2-DSI) project [1], creating the first nodes of an experimental testbed for logistical network that we call the Logistical Backbone (L-Bone). 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.

The logistical networking capabilities supported on this infrastructure will include NWS sensing [18] of the storage and network resources, IBP caches, and state management for IBP Mail inclusions.

A. The Implementation

In its current stage, it is based on a LDAP directory of IBP depots, and informations such as network proximity to any Internet access point can be calculated in real-time. The L-Bone client library allows users to query for depots that satisfy user-specified criteria, such as available space, network or geographical proximity. The server replies with a depots set, the size of which can be chosen by the user in his request.

V. Experience and Applications

A. e-Toile

The e-Toile project [5] aims to build a grid for experimental applications. This grid is intended to be modular: it is built by dedicated servers, provisioned by the partners, servers that might be added to the e-Toile grid permanently or just temporarely. Other servers and other partners can be freely added to the original topology.

After a set-up time, when the Globus [2] middleware has been intensively used, this project aims to explore new middlewares and new concepts for the grid, such as active networking and logistical networking, with the deployment of both active routers, able to run services, and IBP depots, to value the local environment and to allow easy integration between purposely made and more generic tools.

B. Tamanoir

Tamanoir [14] is a project developed by the RESO team of the Ecole Normale Sup/'erieure of Lyon, France, in the in the field of Active Networking. It is composed by a complete framework that allows users to easily deploy and maintain distributed active routers on wide area networks. IBP is not only a part of the set of distributed tools provided, such as routing manager and stream monitoring tool, but it is currently used as a caching tool to improve performance [9]. Any Tamanoir Active Node (TAN) is able to process the data according to a certain service; when the service is not available, the receiver sends a message to the sender asking to provide the service needed. After the deployment of the service, the data can be treated. In such a situation, the IBP depot stores the data while the service is not active yet, therefore improving performance by avoiding the retransmission of the first packets.

IBP depots are also used by a Tamanoir node in a reliable multicast situation. There are three major benefits of performing storage in the network for a reliable multicast application. First of all, we can look at the TAN with its depot as a kind of mirror for data distribution, to download them from the geographically (or network) closest point to the consumers. Another advantage is that clients can consume data with their own processing speed capabilities without disturbing the server where data come from, and finally, a TAN can retransmit lost data without uselessly overloading the network between the server and the TAN.

C. IBP-mail

IBP-Mail [13] is a project developed by the University of Tennessee that uses IBP to transmit and deliver mail attachments that require storage resources beyond the capacity of standard mail servers. After successfully testing its potential with a prototype, we are now focusing on a more robust and scalable architecture. IBP-mail allows a mail attachment to be passed between users by storing it first into a suitable IBP server; then, the Sender forwards the exNode holding the capabilities to the Receiver in a MIME attachment. Upon receiving the attachment, the receiver user'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 the receiver the management capability, and having him deallocate the file. Using the information provided by the L-Bone, a very simple form of file routing has been already implemented in IBP-Mail, allowing the sender to insert the file into an IBP buffer on a depot close to his system, and moving the buffer asynchronously to an IBP buffer close to the receiver, and therefore allowing fast insertion for the sender and fast delivery to the receiver.

D. Netsolve

NetSolve [12] is a widely known project whose aim is to provide remote access to computational resources, both hardware and software. When implementing distributed computation in a wide area network, data can be produced at any location and consumed at any other, and it might be difficult to find the ideal location for the producer of the data, its consumer, and the buffer where the data are stored. To implement a system where globally distributed caches cooperate to move data near consuming resources, IBP was chosen as a natural solution. IBP is now integrated in Netsolve since the version 1.4 (august 2001), and results from testing and normal use show a much-improved efficiency.

VI. Related Works

IBP occupies an architectural niche similar to network file systems such as AFS[16] and Network Attached Storage appliances, but its model of storage is more primitive, making it similar in some ways to Storage Area Networking (SAN) technologies developed for local networks. In the Grid community, projects

such as GASS [11] and the SDSC Storage Resource Broker [6] are file system overlays that implement a uniform file access interface and also impose uniform directory, authentication and access control frameworks on their users.

VII. Conclusion and future works

While some ways of engineering for resource sharing focus on optimizing the use of scarce resources within selected communities, the exponential growth in all areas of computing resources has created the opportunity to explore a different problem, viz. designing new architectures that can take more meaningful advantage of this bounty. The approach presented in this paper is based on the Internet model of resource sharing and represents one general way of using the rising flood of storage resources to create a common distributed infrastructure that can share the growing surplus of storage in a way analogous to the way the current network shares communication bandwidth. It uses the Internet Backplane Protocol (IBP), which is designed on the model of IP, to allow storage resources to be shared by users and applications in a way that is as open and as easy to use as possible while maintaining a necessary minimum of security and protection from abuse. IBP lays the foundation for the intermediate resource management components, accessible to every end-system, which must be introduced to govern the way that applications access and utilise this common pool in a fully storage-enabled Internet

References

[1] http://dsi.internet2.edu.

[2] http://www.globus.org.

[3] http://www.gridforum.org.

[4] http://www.ietf.org.

[5] http://www.urec.cnrs.fr/etoile.

[6] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Ressource Broker. In *CASCON'98*, Toronto, Canada, 1998.

[7] A. Bassi, M. Beck, J. Plank, and R. Wolski. The internet backplane protocol: Api 1.0. Technical Report ut-cs-01-455, University of Tennessee, 2001.

[8] A. Bassi, M. Beck G. Fagg, T. Moore, J. Plank, M. Swany, and R. Wolski. The internet backplane protocol: A study in resource sharing. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE/ACM, may 2002.

[9] A. Bassi, J.-P. Gelas, and L. Lefèvre. Tamanoir-ibp: Adding storage to active networks. In *Active Middleware Services*, pages 27–34, Edinburgh, Scotland, July 2002. IEEE computer society. ISBN: 0-7695-1721-8.

[10] M. Beck, T. Moore, and J. Plank. An end-to-end approach to globally scalable network storage. In *ACM SIGCOMM 20002 Conference, Pittsburgh, PA, USA*, August 2002.

[11] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. Gass: A data movement and access service for wide area computing systems. In *Sixth Workshop on I/O in Parallel and Distributed Systems*, may 1999.

[12] H. Casanova and J. Dongarra. Applying netsolve's network enabled server. *IEEE Computational Science and Engineering*, 5(3), 1998.

[13] W. Elwasif, J. Plank, M. Beck, and R. Wolski. Ibp-mail: Controlled delivery of large mail files. In *NetStore 99, Seattle, WA, USA*, 1999.

[14] Jean-Patrick Gelas and Laurent Lefèvre. Tamanoir: A high performance active network framework. In C. S. Raghavendra S. Hariri, C. A. Lee, editor, *Active Middleware Services, Ninth IEEE International Symposium on High Performance Distributed Computing*, pages 105–114, Pittsburgh, Pennsylvania, USA, August 2000. Kluwer Academic Publishers. ISBN 0-7923-7973-X.

[15] H. Hulen, O. Graf, K. Fitzgerald, and R. Watson. Storage area network and the high performance storage system. In *Tenth NASA Goddard Conference on Mass Storage Systems*, April 2002.

[16] J.H. Morris, M. Satyanarayan, M.H. Conner, J.H. Howard, D.S.H. Rosenthal, and F.D. Smith. Andrew: A Distributed Personal Computing Environment. *Communication of the ACM*, 29(3):184–201, 1986.

[17] D. Patterson, G. Gibson, and M. Satyanarayanan. A case for redundant arrays of inexpensive disks(raid). In *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD), Chicago, IL, USA*, pages 81–94, June 1988.

[18] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In IEEE Press, editor, *6th IEEE Symp. on High Performance Distributed Computing, Portland, Oregon*, 1997.