The Internet Backplane Protocol: A Study in Resource Sharing

Alessandro Bassi, Micah Beck, Graham Fagg, Terry Moore, James S. Plank
Department of Computer Science
University of Tennessee
{abassi,mbeck,fagg,tmoore,plank}@cs.utk.edu

Martin Swany, Rich Wolski

Department of Computer Science
University of California Santa Barbara
{swany,rich}@cs.utk.edu

Abstract

In this work we present the Internet Backplane Protocol (IBP), a middleware created to allow the sharing of storage resources, implemented as part of the network fabric. IBP allows an application to control intermediate data staging operations explicitly. As IBP follows a very simple philosophy, very similar to the Internet Protocol, and the resulting semantic might be too weak for some applications, we introduce the exNode, a data structure that aggregates storage allocations on the Internet.

1. Introduction

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

Interoperability and *scalability* are necessary design goals for distributed systems based on resource sharing, but the two models we consider differ in the positions they take along the continuum between total *control* and complete *openness*. The difference affects the tradeoffs they tend to make in fulfilling their other design goals. The Computer Center model, which came to maturity with the NSF Supercomputer Centers of the 80s and 90s, was developed in order to allow scarce and extremely valuable resources to be shared by a select community in an environment where security and accountability are major concerns. The form of sharing it implements is necessarily highly controlled – authentication and access control are its characteristic design issues. In the last few years this approach has given rise to a resource sharing paradigm known as information technology "Grids." Grids are designed to flexibly aggregate various types of highly distributed resources into unified platforms on which a wide range of "virtual organizations" can build. [1] By contrast, the Internet paradigm, which was developed over the same 30-year period, seeks to share network bandwidth for the purpose of universal communication among an international community of indefinite size. It uses lightweight allocation of network links via packet routing in a public infrastructure to create a system that is designed to be open and easy to use, both in the sense of giving easy access to a basic set of network services and of allowing easy addition of privately provisioned resources to the public

This work is supported by the National Science Foundation Next Generation Software Program under grant # EIA-9975015, the Department of Energy Scientific Discovery through Advanced Computing Program under grant # DE-FC02-01ER25465, and by the National Science Foundation Internet Technologies Program under grant # ANI-9980203.

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

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

2. Background: The Internet Protocol and the Internet Backplane Protocol

The Internet Backplane Protocol is a mechanism developed for the purpose of sharing storage resources across networks ranging from rack-mounted clusters in a single machine room to global networks [2-4]. To approximate the openness of the Internet paradigm for the case of storage, the design of IBP parallels key aspects of the design of IP, in particular IP datagram delivery. This service is based on packet delivery at the link level, but with more powerful and abstract features that allow it to scale globally. Its leading feature is the independence of IP datagrams from the attributes of the particular link layer, which is established as follows:

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

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

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

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

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

- Aggregation of access layer blocks masks the fixed block size;
- Fault detection with a very simple failure model (faulty byte arrays are discarded) masks the variety of different failure modes;
- Global addressing based on global IP addresses masks the difference between access layer addressing schemes.

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

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

Regardless of the strengths of the IP paradigm, however, its application here leads directly to two problems. First, in the case of storage, the chronic vulnerability of IP networks to Denial of Use (DoU) attacks is greatly amplified. The free sharing of

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

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

Both issues are addressed through special characteristics of the way IBP allocates storage:

- *Allocations of storage in IBP can be time limited.* When the lease on an allocation expires, the storage resource can be reused and all data structures associated with it can be deleted. An IBP allocation can be refused by a storage resource in response to over-allocation, much as routers can drop packets, and such "admission decisions" can be based on both size and duration. Forcing time limits puts transience into storage allocation, giving it some of the fluidity of datagram delivery.
- *The semantics of IBP storage allocation are weaker than the typical storage service.* Chosen to model storage accessed over the network, it is assumed that an IBP storage resource can be transiently unavailable. Since the user of remote storage resources is depending on so many uncontrolled remote variables, it may be necessary to assume that storage can be permanently lost. Thus, *IBP is a "best effort" storage service.* To encourage the sharing of idle resources, IBP even supports "volatile" storage allocation semantics, where allocated storage can be revoked at any time. In all cases, such weak semantics mean that the level of service must be characterized statistically.

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

3. The IBP Service, Client API, and Current Software

IBP storage resources are managed by "depots," or servers, on which clients perform remote storage operations. As shown in Table 1, the IBP client calls fall into three different groups:

Storage Management	Data Transfer	Depot Management

`IBP_allocate`
`IBP_manage`
	`IBP_store`
	`IBP_load`
	`IBP_copy`
	`IBP_mcopy`
		`IBP_status`

Table 1: IBP client calls

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

IBP_copy and IBP_mcopy calls provide data transfer, and will be analyzed in more depth in section 4.3. A more detailed account of the API and its other functions is available online (http://icl.cs.utk.edu/ibp/documents).

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

4. IBP Design Issues

A design for shared network storage technology that draws on the strengths of the Internet model must also cope with its weaknesses and limitations. In this section we sketch the key design issues that we have encountered in developing IBP.

4.1 Security

The basis of IBP network security is the capability. Capabilities are created by the depot in response to an allocation request and returned to the client in the form of long, cryptographically secure byte strings. Every subsequent request for actions on the allocated byte array must then present the appropriate capability. As long as capabilities are transmitted securely between client and server, and the security of the depot itself is not compromised, only someone who has obtained the capability from the client can perform operations on the byte array. It is important to notice that this is the only level of security that IBP must deal with: the data encryption, because of its end-to-end nature, has to be handled in the layer(s) above IBP. For example, the IBP-mail application (see section 6.4) encrypts all the data sent, but this is, and has to be, completely transparent to the IBP depot. The same applies for data compression and any other end-to-end service.

4.2 Timeouts

Timeouts are implemented in both the IBP library and server for different purposes. The client library deals with the possibility that faults in the network or server will cause a particular request to hang. On the server side, the fact that reading and writing to a byte array may occur simultaneously means that a particular IBP_load or IBP_store operation may not be able to fulfill its byte count immediately; to avoid such problems, a server timeout (or Server Sync) has been implemented to let the server wait a specified amount of time for the conditions requested to be available before answering a particular request.

4.3 Data mover

Since the primary intent of IBP is to provide a common abstraction of storage, it is arguable that third party transfer is unnecessary. Indeed, it is logically possible to build an external service for moving data between depots that access IBP allocations using only the IBP_load and IBP_store; however, such a service would have to act as a proxy for clients, and this immediately raises trust and performance concerns. The IBP_copy and IBP_mcopy data movement calls were provided in order to allow a simple implementation to avoid these concerns, even if software architectures based on external data movement operation are still of great interest to us [5].

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

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

4.4 Depot Allocation Policy and Client Allocation Strategies

One of the key elements of the IBP design is the ability of depots to refuse allocations and the strategies used by clients in the face of such refusal. IBP depots implement an allocation policy based on both time and space. Any allocation has to be lower

than the curve defined by "space*time = K", where K varies linearly with the remaining free space. This permits smaller allocations to have a longer duration. This policy gives us some defense to DoU attacks, because any allocation of storage that depletes a large fraction of the remaining space in a particular depot will tend to be limited to a short duration, thereby limiting the impact on other depot users.

5. The exNode

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

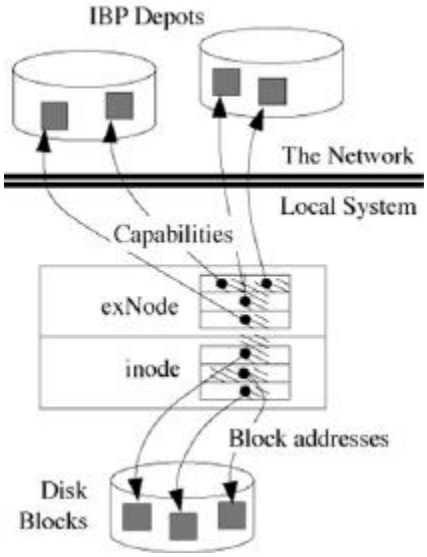

Figure 1: Inode compared to exNode – The exNode implements a file turned inside out.

Following the example of the Unix inode, which aggregates disk blocks to implement a file, we have chosen to implement a single generalized data structure, which we call an *external node,* or *exNode,* in order to manage aggregate allocations that can be used in implementing network storage with many different strong semantic properties. Rather than aggregating blocks on a single disk volume, the exNode aggregates storage allocations on the Internet, and the exposed nature of IBP makes IBP byte-arrays exceptionally well adapted to such aggregations (Figure 1). In the present context the key point about the design of the exNode is that it has allowed us to create an abstraction of a network file to layer over IBP-based storage in a way that is completely consistent with the exposed resource approach.

The exNode library has a set of calls (Table 2) that allow an application to create and destroy an exNode, to add or delete a mapping from it, to query it with a set of criteria, and to produce an XML serialization, to allow the use of XML-based tools and interoperability with XML systems and protocols.

exNode Management	**Segment Management**

`XndCreateExNode`
`xndFreeExNode`
	`xndCreateSegment`
	`xndFreeSegment`
	`xndAppendSegment`
	`xndDeleteSegment`

Segment Query	**exNode Serialization**

`XndQuery`
`xndEnumNext`
`xndFreeEnum`
	`XndSerialize`
	`xndDeserialize`

Table 2: exNode Library calls

The software is currently in its final alpha release and we plan to make it available together with our IBP distribution in the very near future.

6. Applications

6.1 Data Caching in NetSolve

NetSolve 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. The experiment conducted by the NetSolve group placed a Client application at the University of California, San Diego, that requested a pool of computational resources and data storage at the University of Tennessee. The results [6-8] are encouraging, showing a much-improved efficiency.

6.2 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)*. Following an approach that is similar to that proposed by Salehi *et al* for use in multicast video transmission [9], 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. Results from experiments with early LSL prototypes are impressive [10], and we expect similar results from the IBP implementation.

6.3 Parallel I/O for distributed applications

The MPI_connect package [11] 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 can be executed at a number of distributed sites. One feature of the MetaQueuing system is that the user is not aware of where the parallel application will execute until runtime. This has the side effect that any large input files required will 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 and retrieving the necessary files. This avoids the need to replicate files or to use a complex distributed file system such as AFS, while still allowing applications to access globally distributed data.

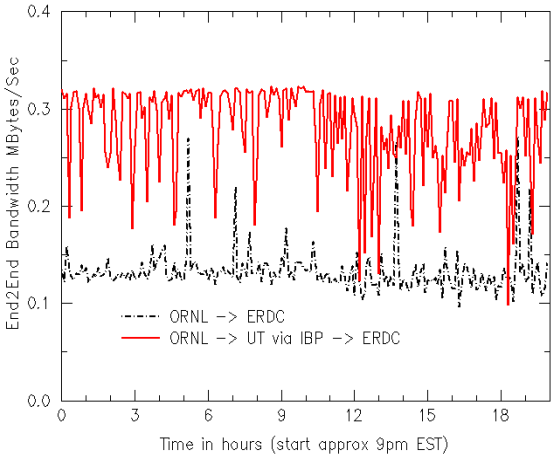

Figure 2- IBP for Parallel I/O

An important feature of this system is its ability to move very large files in the most efficient method via a number of possible paths to avoid network bottlenecks. Figure 2 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 A-B is poor compared to the connectivity of the other paths (A-C, B-C). Thus, MPI_Conn_IO automatically uses 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 in Figure 1 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 per hour. As can the graph makes clear, the use of IBP improves performance by over a factor of two when transferring large files.

6.4 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. 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 capability, which will be replaced soon by exNode, 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 already been 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. This allows for fast insertion by the sender and fast delivery to the receiver.

7. Related Work

IBP occupies an architectural niche similar to that of network file systems such as AFS [12] and Network Attached Storage appliances [13], but its model of storage is more primitive, making it similar in some ways to Storage Area Networking (SAN) technologies developed for local networks. In the Grid community, projects such as GASS [14] and the SDSC Storage Resource Broker [15] are file system overlays that implement a uniform file access interface and also impose uniform directory, authentication, and access control frameworks on their users.

8. Conclusions

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

9. Bibliography

1. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of SuperComputer Applications*, vol. 15, no. 3, 2001.

2. J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski, "The Internet Backplane Protocol: Storage in the Network," presented at NetStore99: The Network Storage Symposium, Seattle, WA, 1999.

3. M. Beck, T. Moore, J. Plank, and M. Swany, "Logistical Networking: Sharing More Than the Wires," in *Active Middleware Services*, vol. 583, *The Kluwer International Series in Engineering and Computer Science*, S. Hariri, C. Lee, and C. Raghavendra, Eds. Boston: Kluwer Academic Publishers, 2000.

4. A. Bassi, M. Beck, J. Plank, and R. Wolski, "Internet Backplane Protocol: API 1.0," Department of Computer Science, University of Tennessee, Knoxville, TN, CS Technical Report, ut-cs-01-455, March 16, 2001, 2001.

5. M. Beck, T. Moore, and J. S. Plank, "Exposed vs. Encapsulated Approaches to Grid Service Architecture," presented at 2nd International Workshop on Grid Computing, Denver, CO, Nov. 12, 2001.

6. D. C. Arnold, S. S. Vahdiyar, and J. Dongarra, "On the Convergence of Computational and Data Grids," *Parallel Processing Letters*, vol. 11, no. 2, pp. 187-202, 2001.

7. D. C. Arnold, D. Bachmannd, and J. Dongarra, "Request Sequencing: Optimizing Communication for the Grid," in *Euro-Par 2000 -- Parallel Processing, 6th International Euro-Par Conference*, vol. 1900, *Lecture Notes in Computer Science*, A. Bode, T. Ludwig, W. Karl, and R. Wismuller, Eds. Munich: Springer Verlag, 2000, pp. 1213-1222.

8. H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments,"

presented at 9th Heterogeneous Computing Workshop (HCW'00), May 2000, 2000.

9. J. D. Salehi, Z. L. Zhang, J. F. Kurose, and D. Towsley, "Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing," presented at ACM SIGMETRICS, Philadelphia, PA, May 1996, 1996.

10. M. Swany and R. Wolski, "Data Logistics in Networking: The Logistical Session Layer," presented at submitted to High Performance Distributed Computing, 2001.

11. G. E. Fagg, K. S. London, and Jack J Dongarra, "MPI_Connect, Managing Heterogeneous MPI Application Interoperation and Process Control," in *Lecture Notes in Computer Science*, vol. 1497: Springer Verlag, 1998., pp. 93-96.

12. J. H. Morris, M. Satyanarayan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, and F. D. Smith, "Andrew: A Distributed Personal Computing Environment," *Communications of the ACM*, vol. 29, no. 3, pp. 184-201, 1986.

13. G. Gibson and R. V. Meter, "Network Attached Storage Architecture," *Communications of the ACM*, vol. 43, no. 11, pp. 37-45, 2000.

14. J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," presented at Sixth Workshop on I/O in Parallel and Distributed Systems, May 5, 1999, 1999.

15. C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC Storage Resource Broker," presented at CASCON'98, Toronto, Canada, 1998.