Optimizing Performance and Reliability in Distributed Computing Systems Through Wide Spectrum Storage

James S. Plank♠ Micah Beck♠ Jack Dongarra♠ Rich Wolski♡
Henri Casanova♢ *

Abstract

In this paper, we provide an overview of **Logistical Runtime System (LoRS)**. LoRS is an integrated ensemble of tools and services that aggregate primitive (best effort, faulty) storage allocations to obtain strong properties such as capacity, performance, reliability, that Grid applications desire. The paper focuses on the design and implementation of LoRS, as well as the *storage scheduling* decisions that LoRS must make.

1 Introduction

Over the past two decades the exponential growth in available computing resources has inspired and emboldened evangelists within the research community to tout the revolutionary potential of high performance distributed computing. The idea of an increasingly rich and rapidly spreading resource fabric, replete with high bandwidth network links, high performance processing, and massive storage, has been a powerful stimulus to the development of advanced distributed infrastructures – most notably "the Grid" [12] – for advanced scientific applications. Conventional approaches [11, 15, 19] to building a Grid assume an existing *fabric* layer of networking, storage and computational resources [13] on top of which wide area systems are built, in the tradition of *metacomputing* [22]. In this context what distinguishes our research program in Logistical Computing and Internetworking (LoCI), and the work on Logistical Networking [5, 18] at its core, is our insistence that the Grid fabric itself must be rethought and rearchitected for performance, flexibility and scalability. Our guides in that work are two of the most enduring examples of successful system architecture: the Internet and the Unix file system.

Logistical Networking, the core of our proposal, promotes an unconventional model of network storage management because it takes the view that storage can be used to augment data transmission as part of a unified network resource framework, rather than simply being a network-attached resource. The adjective "logistical" is meant to evoke an analogy with military and industrial networks for the movement of material which requires the coscheduling of long haul transportation, storage depots and local transportation as coordinate elements of a single infrastructure.

We have designed and implemented a basic mechanism, called the *Internet Backplane Protocol (IBP)* [18], that makes it possible to create a storage fabric fulfilling these conditions, and we have used this technology to achieve results that we believe are compelling. But such a storage fabric, while essential, is only a first step toward providing an adequate foundation for the data logistics of Grid applications. It is clear that large scientific simulations and other advanced applications frequently require storage services that provide complex functionality with very strong properties, for example huge capacity, high reliability, guaranteed access speed, and indefinite duration. However, IBP be design is too primitive to be used easily in such a way.

Our work on the design of Logistical Networking has been "from the bottom up," revolving around the concept of a **Network Storage Stack** (Figure 1) . The device itself is a *physical* layer, and the operating system models a *data access* layer on top of that. IBP implements an *storage network* layer, providing a general, interoperable view of scalable "best effort" network storage. On top of IBP we have built the *exNode*, a data structure to aggregate storage allocations and represent complex configurations of data scattered throughout the network. Having reached this point, we have at our disposal a powerful set of tools for implementing storage allocation and data scheduling decisions. The important question is *how* such decisions are made.

In this paper, we provide an overview of **Logistical Runtime System (LoRS)**. LoRS is an integrated ensemble of tools and services that aggregate primitive IBP

* ♠ Department of Computer Science, University of Tennessee. ♡ Department of Computer Science, University of California at Santa Barbara. ♢ Department of Computer Science, University of Tennessee. University of California at San Diego. This material is based upon work supported by the National Science Foundation under grants ACI-0204007, EIA-0224441, ANI-0222945, ANI-9980203, EIA-9972889, and EIA-9975015, and the Department of Energy under grant DE-FC02-01ER25465.

storage allocations to obtain the strong properties described above that Grid applications desire. There are two facets to LoRS research and development. The first focuses on software infrastructure. Techniques for aggregating storage (caching, striping, replicating, etc) are well-known from the arenas of databases, operating systems and parallel programming. We have built LoRS based on these techniques, and will detail the design decisions that we have made.

The second facet answers the following question: *Can we effectively aggregate an imperfectly characterized collection of network storage resources in such as way as to make strong guarantees of their combined performance to higher-level applications?* Typically, work on resource aggregation assumes that the resources available are very restricted, for example the disks installed on a specific RAID storage system, or a file cache residing on an end-user's system in a local area network. As such, performance models and policies have rather strong guarantees about the properties of the components. Because IBP models global storage resource interoperably, decisions regarding choice of storage resources (in other words, *storage scheduling*) can make use of a potentially huge, unstructured pool of storage resources. Obtaining particular aggregate characteristics of subsets of this pool means working backwards to determine the necessary characteristics of the individual storage resources used. Applying optimal scheduling algorithms to the entire set of globally available storage resources is likely to be intractable, so we focus instead on restricting the scope of resources considered and applying algorithms based on a mix of theory and experience, much as has been the case when implementing global routing in the Internet.

We believe that the success of this work has created a technology that gives Grid application developers unparalleled power to manage data and application state in the wide area, using a shared storage resource fabric that can be massively provisioned for that purpose.

2 Storage Fabric for the Grid: The Network Storage Stack

The idea of the "Grid fabric layer," which we also refer to as the "resource fabric," is a product of the effort of today's Grid architects to provide a blueprint and rationale for the Grid's overall design concept [13]. Seen as analogous to the Link layer of the Internet protocol stack, the Grid fabric contains all the diverse networking, computing and storage resources that serve as the underlying foundation for the Grid edifice. The Grid builds on the resource fabric layer just as the IP network builds on diverse Link layer resources. However, unlike

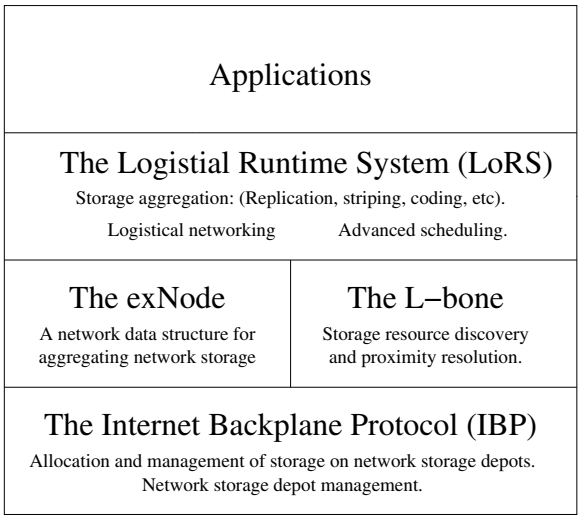

Figure 1: The Network Storage Stack

the architects of the network stack, who make a science and an art of building thin layers at the bottom of the stack, the current architecture of the Grid assumes that the Grid fabric consists of complex systems with failure modes and performance characteristics that are difficult to model and predict. In the case of storage, this means using reliable, high performance systems with redundancy (e.g. RAID) and caching implemented in a single network, and exporting a file or database interface.

However, the end-to-end argument in networking [21], which applies also to network storage [4], tells us that building strong properties and high level abstractions into local systems does not assure these same properties in the wide area: global management is necessary. When global management is needed, local management can be either wasteful or even counterproductive. Similarly, the experience of distributed system builders shows that network resources must be treated in an inherently different manner from local resources; although a uniform *model* of resources may exist that spans local and wide-area settings, a uniform *method of usage* will not [24]. The prevailing approach to Grid architecture accepts file and database abstractions as the fundamental models for storage in the resource fabric, and therefore must live with the problems this generates.

For our work in Logistical Networking we have defined a different basic abstraction for the storage resource fabric and are in the process of building a *Network Storage Stack*, diagrammed in Figure 1. Note that this structure jettisons the well-known methods of usage developed for local-area storage, viz. file systems, databases, VM mapping. This seemingly radical choice was determined by the desire to create a bottom-up, lay-

ered design for the network storage stack that adheres to the same the end-to-end principles used in the design of the Internet.

2.1 IBP: The Internet Backplane Protocol

The lowest level above basic devices and operating systems is the Internet Backplane Protocol (IBP). IBP is a very basic mechanism for clients to allocate and use storage located at storage depots on the network. The goal of IBP is network transparency, i.e. it comes as close as it possibly can to sharing bare storage on the network. To achieve such transparency it supplies an abstraction of *access layer resources* (i.e. storage services at the local level) that does at least the following two things:

- Exposes underlying storage resources in order to maximize freedom at higher levels.
- Enables scalable Internet-style resource sharing.

This higher level of abstraction allows a uniform IBP model to be applied to storage resources globally, which is essential to creating the most important difference between access layer block storage and IBP byte array service: *Any participant in an IBP network can make use of any access layer storage resource in the network regardless of who owns it.* The use of IP networking to access IBP storage resources creates a global storage service.

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

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

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

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

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

As already mentioned, the allocation function is the most important element. `IBP_allocate()` is used to allocate a byte array at an IBP depot, specifying the size, duration (permanent or time limited) and other attributes. A chief design feature is the use of text-based

capabilities (cryptographically secure passwords). A successful allocation returns a set of three capabilities: one for reading, one for writing, and one for management of the allocated byte array. A more detailed account of the API and its other functions, as well as a description of the status of the current software the implements that IBP client, servers, and protocol is available at `http://loci.cs.utk.edu`.

2.2 The exNode

In order to create interoperable services based on aggregate IBP storage resources a standard container is needed for the creation and exchange of complex aggregate structures. We view the *exNode* as the first and most important, but not necessarily the only such data structure.

In our exposed-resource paradigm, implementing abstractions with strong properties — reliability, fast access, unbounded allocation, unbounded duration, etc. — involves creating a construct at a higher layer that *aggregates* more primitive IBP byte-arrays below it, often distributed at multiple locations. For example, caching requires that data be held in a home site, but temporary copies are made at various remote sites. Similarly, replication requires that multiple copies of data exist in various locations for purposes of performance and fault-tolerance. More advanced logistical applications require that data be explicitly routed through the network, and thus may have many "homes" throughout their lifetime.

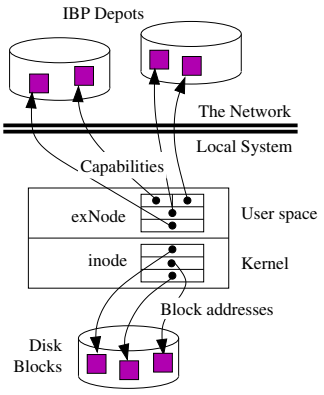

Figure 2: The exNode in comparison to the Unix inode

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 there is 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 [7]) or through modifications to the file system or additional software layers that make redundant allocations and maintain additional state (e.g. AFS [16], HPSS [25]).

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 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 buffers in IBP depots to form something like a file, with the buffers acting as disk blocks. Two major differences between exNodes and inodes are that the IBP buffers may be of any size, and the extents may overlap and be replicated. In the present context, the key point about the design of the exNode is that it allows us to create storage abstractions with stronger properties, such as a network file, which can be layered over IBP-based storage in a way that is completely consistent with the exposed resource approach.

Since our intent is to use the exNode file abstraction in a number of different applications, we have chosen to express the exNode concretely as an encoding of storage resources (e.g. IBP capabilities) and associated metadata in XML. Like IBP capabilities, these serializations may be passed from client to client, allowing a great degree of flexibility and sharing of network storage. If the exNode is placed in a directory, the file it implements can be imbedded in a namespace. But if the exNode is sent as a mail attachment, there need not be a canonical location for it. The use of the exNode by varying applications 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 the storage resources that constitute the data component of the file's state: (1) The portion of the file extent implemented by a particular resource (starting offset and ending offset in bytes), (2) the service attributes of each constituent storage resource (e.g. reliability and performance metrics, duration), and (3) the total set of storage resources which implement the file and the aggregating function (e.g. simple union, parity storage scheme).

2.2.1 The L-Bone

The L-Bone (Logistical Backbone) is a distributed runtime layer that allows clients to perform IBP depot discovery. IBP depots register themselves with the L-Bone, and clients may then query the L-Bone for depots that have various characteristics, including minimum storage capacity and duration requirements, and basic proximity requirements. Once the client has a list of IBP depots, it may then request that the L-Bone use the Network Weather Service (NWS) [26] to order those depots according to bandwidth predictions using live networking data. Thus, while IBP gives clients access to remote storage resources, it has no features to aid the client in figuring out which storage resources to employ. The L-Bone's job is to provide clients with those features.

Currently, the L-Bone is written on top of LDAP, and supports the functionalities described above. Since it is based on LDAP, L-Bone functionality could also be provided by other similar services, such as the Globus Metadirectory Service (MDS) [8], and indeed we anticipate MDS will provide users with information about IBP resources as those resources become more widely deployed and more heavily provisioned. The L-Bone is currently composed of 143 depots at locations in the United States, Europe, Asia, and Australia, serving over ten terabytes of network storage. This number will continue to grow as more users participate in our projects. On the basis of ongoing collaborations between the PI's and various segments of the Grid community, we anticipate broad deployment and heavy provisioning of an IBP-based storage resource fabric across Planet Lab, ESNET, Internet2, and the PACI community. L-Bone (or MDS) services will provide basic information services for this resource fabric. For the L-Bone API, and its up-to-date composition, see `http://loci.cs.utk.edu/lbone`.

3 The Logistical Runtime System

Our work with IBP has demonstrated conclusively that when applications can allocate and use time-limited, secure storage from network storage depots, they can dramatically improve their performance and functionality [10, 6, 1, 9, 2]. These demonstrations have validated our research agenda, and have spurred us to continue research in this arena. Each of these applications, however, was carefully programmed to implement the customized storage management functions it needed. Taken as a whole, they both demonstrate the need for, and motivate the design of higher level tools, abstractions and services for network storage.

We term this combination of tools, abstractions, and methodologies with a common, high-performance infrastructure for implementing them the *Logistical Runtime System* (LoRS). Our goal in developing LoRS is to ease the programming burden associated with building Grid applications by encapsulating the strategies that are necessary to implement useful storage functions (e.g. replication, caching, aggregation) within a set of runtime system services. At the same time, the storage service architecture must be extensible so that new abstractions may be incorporated easily. As such, LoRS will serve as a mechanism for generalizing the customized storage strategies that many Grid applications now implement and re-implement on a case-by-case basis.

The importance of this generalization capability cannot be overstated for the Grid. At present, most Grid applications are customized, one-time-only solutions that are difficult to develop. While systems such as the Grid Information System [8] are attempting to generalize information management functions, we know of no other effort that is focusing on Grid storage abstractions. The LoRS will serve as a model for and a focal point for this important aspect of high-performance distributed computing.

3.1 LoRS Tools

A simple, yet useful part of the Logistical Runtime System is a set of tools that make it easy to employ and aggregate network storage in the form of IBP depots in robust and powerful ways. Again, these tools build on the lower levels of the network storage stack (Figure 1): IBP is the basic storage substrate, serving time-limited storage buffers to its clients. The L-Bone is employed to find IBP depots matching various properties (size, duration, network proximity), and to do live proximity detection measurements using the Network Weather Service. The exNode is the data structure for storing data, that may be serialized with XML so that it may be passed around the network.

The tools are as follows:

LoRS_upload: This tool uploads a local file into the network and returns an exNode for the upload. This upload may be parameterized in a variety of ways. For example, the user may partition the file into multiple blocks (i.e. stripe the file) and these blocks may be replicated on multiple IBP servers for fault-tolerance and/or proximity reasons. Moreover, the user may specify proximity metrics for the upload, so the blocks have a certain network location.

LoRS_download: This takes an exNode as input, and downloads the file that it represents into a local file. This involves coalescing the replicated fragments of the file, and must deal with the fact that some fragments may be closer to the client than others, and some may not be available (due to time limits, volatility, and

standard network failures). **LoRS_download** may only download portions of the file, and if desired, it can operate in a streaming fashion, so that the client only has to consume small, discrete portions at a time.

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

LoRS_augment: This takes an exNode as input, adds more replicas to it (or to parts of it), and returns an updated exNode. Like **LoRS_upload**, these replicas may have a specified network proximity.

LoRS_trim: This takes an exNode, deletes specified fragments, and returns a new exNode. These fragments may be specified individually, or they may be specified to be those that represent expired IBP allocations. **LoRS_augment** and **LoRS_trim** may be combined to effect a routing of a file from one network location to another — first it is augmented so that it has replicas near the desired location, then it is trimmed so that the old replicas are deleted.

The LoRS tools are much more powerful as tools than IBP capabilities, since they allow users to aggregate storage units in network storage depots. However, the tools have two basic limitations. First, more complex applications require complex interactions between the data represented by exNodes. As such, simple tools will not be enough. Second, although the tools have been designed so that they may encompass a variety of aggregation needs, the actual algorithms, abstractions, mechanisms and policies needed to implement these types of aggregation have been left unspecified. A large amount of our research focuses on these algorithms, abstractions, mechanisms, and policies. We describe this research in the next section.

3.2 LoRS Algorithms

Algorithms for aggregating storage are straightforward, and have seen use in database systems, operating systems, parallel programming systems and checkpointing sytems. The algorithms that we plan to implement as methods for aggregation in the LoRS tools and services are:

- **Capacity**: This involves simply aggregating storage resources to allow users to store files that are bigger than any one storage depot.
- **Caching**: This involves placing temporary copies of data near anticipated downloaders.
- **Striping**: This involves breaking up logical data units (e.g. files) into blocks, and putting the blocks in separate locations so that they may be simultaneously downloaded, either by one downloading client, or multiple downloading clients.
- **Replication**: Replication is like caching, except there is no notion of a "home" copy and a "temporary" copy. The intent of replication is either to have replicas near downloading clients for performance, or to be able to tolerate server/network failures, since clients have a choice of servers from which to download [3]. Obviously, replication can be done on whole data files, or on blocks of data files.
- **Coding**: When files are broken into blocks and striped across multiple servers, coding blocks may be calculated from the data blocks, and then used to recover lost blocks following a failure. Recovery involves calculating the contents of the lost block(s) from the contents of the extant blocks and the coding blocks. Standard coding methods are parity [7, 14] which involves only exclusive-*or*'s of data blocks, and Reed-Solomon coding [17], which can achieve the same level of fault-tolerance with fewer coding blocks.
- **Logistical Routing**: When data needs to move from network location to another, the staging of this data may be done explicitly for optimal control of performance. The power of Logistical Routing has been shown in a mail attachment routing application [10] and in a low-level networking demonstration [23].

3.3 LoRS Abstractions

The simplest storage abstraction is one that is based on a specific homogeneous physical resource, resulting in predictable access characteristics. For example, consider a conventional file system built on a specific disk volume and attached to the I/O bus of one system. The speed, reliability and capacity of each file are functions of the characteristics of the disk and the system to which it is attached, which, by and large, is straightforward to model and reason about [20].

However, this simple view changes when a storage service is built on a collection of heterogeneous resources, with policy being used to determine the how to allocate those resources to meet a set of competing requirements. For example, in the world of single systems:

- A hierarchical file system may be based on a robotic tape library and a disk cache, using policies such as LRU and read-ahead as well as specific user directives to anticipate future requirements.
- A distributed file system may combine the use of a highly reliable centralized server with local caches to provide a shared service.

In these cases, the "file" as an abstraction is difficult to use for modeling, scheduling, or reasoning, because the performance characteristics of a file depend on how and where it is stored in the complex, heterogeneous system.

The abstractions that suit our purposes are based on the exposed nature of our resources. For example, IBP has been designed so that when one allocates an IBP buffer, one may use the knowledge about its location, reliability, and current performance conditions (all as reported by the L-Bone and NWS) to reason about it for scheduling purposes.

In terms of scheduling and aggregation of resources, there are a variety of *dimensions of service* about which we need to reason. For example, to schedule for performance, we need to know about IBP depot performance, network locations, and network conditions. To aggregate for fault-tolerance, we need to know about failure rates. When dealing with large data sets, we need to know about capacity. And so on.

Therefore, the abstractions that we need for scheduling and aggregation of storage resources are the myriad dimensions of services that need to be reasoned about in order to make effective storage allocation decisions. We see these as including, but not being limited to: capacity, base performance, reliability (MTTF), availailability (MTTF/(MTTF+MTTR)), time limits, current proximity to network locations.

Implementationally, these dimensions of service will be reported to scheduling clients by the L-Bone (which employs the NWS for monitoring and prediction).

3.4 LoRS Methodologies

A final component of the Logistical Runtime System is the methodology used to select resources for aggregation. This falls into the category of scheduling. Formally, we can view each resource i as having, at each point in time, a vector r_i of properties. These are static and dynamic measurements along each dimension of service (e.g. capacity, reliability, performance relative to a host, etc). As stated above, r_i can be determined by physical characteristics and by the monitoring/predictive ability of the Network Weather Service, and reported by the L-Bone.

Each application j goes through a variety of scheduling points involving the aggregation of resources. At each of these points, it selects a suite of resources to aggregate according to the LoRS algorithms. The goal is for this aggregation to fulfull baseline metrics along a number of dimensions (e.g. capacity, reliability, performance). These metrics may be represented by a vector a_j. Note, the two vectors defined so far do not have the same dimensionality – the vectors r_i concern *individual properties of a resource*, and the vectors a_j concern *collective properties of an aggregation of resources*.

Each LoFS aggregation algorithm k aggregates resources in a known way, so that when a set S of resources is aggregated, the collective properties of the set may be computed from the individual properties of the resources in the set by a function F_k of the algorithm. For example, given independent failure modes, if the probability of an IBP depot's being functional is 0.99, then replication of data on n depots will yield an overall probability of $(1 - (.01)^n)$ that an application will be able to download the piece of data from one of the replicas.

Therefore, the job of a scheduler is to select a set of resources that, when aggregated according to algorithm k for application j, meets the application's desires. Formally, this is a set S such that:

$$F_k(S) \geq a_j.$$

While the tractability of this approach may be a concern, many of the aggregation functions (F_k) will involve simple mathematical expressions (arithmetic as above, or determining maxima and minima), leading us to believe that in many cases, especially when the application's desires are reasonable to fulfill, we will be able to use this approach to make good scheduling decisions.

4 Conclusions

Our Logistical Computing and Internetworking reserach agenda has reached the point of maturity where it must be driven by the requirements of Grid applications in the field. In the area of storage, the requirements of applications are expressed as broad quality of service requirements covering reliability and performance. The role of the Logistical Runtime System is to address these requirements, parameterized by a small number of well-defined characteristics, and to perform scheduling and management functions to implement them on top of the advanced Grid Fabric.

What is unprecidented in the task that we have set ourselves is the scope of the collection of possible resources that our schedulers can work with. In a Logistical Network, every network that has a router may also have an IBP depot that can be used to store data. The state of those depots will be in constant flux and so must be monitored and predicted. Because our goal is to meet the demanding requirements of high performance Grid applications, we must aggregate "best effort" underlying resources to create a highly engineered storage platform.

The convergence of storage, networking and ultimately computation is often touted by those who champion wide area distributed computing and the Grid. The

slogan that "the network is the computer" is often repeated in this context. If the network is to become our computer, the fabric on which we compute must be more flexible, more open to scheduling of both storage and computation than conventional approaches allow. Logistical Computing and Internetworking is an attempt to rethink the Grid fabric with the intention of making the use of globally distributed resources natural and transparent. Then the era of Grid computing will truly be upon us.

References

[1] A. Agbaria and J. S. Plank. Design, implementation, and performance of checkpointing in NetSolve. In *International Conference on Dependable Systems and Networks (FTCS-30 & DCCA-8)*, pages 49–54, June 2000.

[2] D. C. Arnold, S. S. Vahdiyar, and J. Dongarra. On the convergence of computational and data grids. *Parallel Processing Letters*, 11(2):187–202, 2001.

[3] S. Atchley, S. Soltesz, J. S. Plank, M. Beck, and T. Moore. Fault-tolerance in the network storage stack. In *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, Ft. Lauderdale, FL, April 2002.

[4] M. Beck, T. Moore, and J. S. Plank. An end-to-end approach to globally scalable network storage. In *ACM SIGCOMM '02*, Pittsburgh, August 2002.

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

[6] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the grid. In *SC00 Conference on High-Performance Computing*, November 2000.

[7] P. M. Chen *et al*. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[8] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Procedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, 2001.

[9] W. Elwasif, J. S. Plank, and R. Wolski. Data staging effects in wide area task farming applications. In *IEEE International Symposium on Cluster Computing and the Grid*, pages 122–129, Brisbane, Australia, May 2001.

[10] W. Elwasif *et al*. *IBP-Mail*: Controlled delivery of large mail files. In *NetStore '99: Network Storage Symposium*, October 1999.

[11] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, Summer 1998.

[12] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, July 1998.

[13] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[14] G. A. Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*. The MIT Press, Cambridge, Massachusetts, 1992.

[15] A. S. Grimshaw, W. A. Wulf, and The Legion Team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[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. *Communications of the ACM*, 29(3):184–201, 1986.

[17] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[18] J. S. Plank, A. Bassi, M. Beck, T. Moore, D. M. Swany, and R. Wolski. Managing data storage in the network. *IEEE Internet Computing*, 5(5):50–58, September/October 2001.

[19] J. Pruyne and M. Livny. A worldwide flock of condors : Load sharing among workstation clusters. *Future Generation Computer Systems*, 12, 1996.

[20] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–29, March 1994.

[21] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems,*, 2(4):277–288, November 1984.

[22] L. Smarr and C. E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, 1992.

[23] M. Swany and R. Wolski. Data logistics in network computing: The logistical session layer. In *IEEE International Symposium on Network Computing and Applications*. IEEE, October 2001.

[24] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. Technical Report SMLI TR-94-29, Sun Microsystems, November 1994.

[25] R. W. Watson and R. A. Coyne. The parallel I/O architecture of the high-performance storage system (HPSS). In *IEEE Mass Storage Systems Symposium*, 1995.

[26] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, 1999.