NeST: Network Storage Technologies
DRAFT White Paper *

John Bent, Venkateshwaran V
Department of Computer Science, University of Wisconsin

July 2001

*Please see www.nestproject.com for more information.

Abstract

Current storage appliances have been traditionally designed to meet either the storage demands of a local or a wide area network. The grid is unique in that it escapes this limitation and allows users to share resources both within and between local area networks. However, this freedom creates many new challenges to storage appliances that would be used on the grid.

NeST is a user-level software-only storage appliance that is being specifically designed to meet the unique storage needs of grid computation. A virtual protocol layer allows arbitrary data transfer and authentication protocols to be used. An abstract storage interface allows a wide range of physical storage devices to be optimally utilized. Multiple concurrency architectures provide the flex needed to achieve high levels of throughput across multiple host platforms. Flexible mechanisms allow storage within NeST to be reserved and there is support for differentiated levels of service as well. Finally, a policy language such as Condor ClassAds can be used to advertise the NeST and allow discovery of both the physical storage and of the data contained thereon.

1 Introduction

An appliance is any tool that performs one function well. The general definition of an appliance is well-understood and has already been thoroughly discussed [5]. Users expect that appliances are easy to use and are both robust and reliable.

Storage appliances, more specifically, have traditionally been thought of as the tools used to store and retrieve objects from storage[1]. The need for reliable, scalable, manageable, and high performing network storage has led to a proliferation of commercial storage appliances by vendors such as NetApp [6] and EMC. [3]

However, there are several limitations with the current storage systems offered today. Not all current storage systems suffer from all of these limitations but there are none that adequately address all. In short, current storage solutions were designed either for wide area or local area storage; none of their designs seem guided by practical considerations of the nature of the grid.

2 Grid storage challenges

Current storage appliances have been traditionally designed to meet either the storage demands of a local or a wide area network. The grid [4] is unique in that it escapes this limitation and allows users to share resources both within and between local area networks. However, this freedom creates many new challenges to storage appliances that would be used on the grid.

2.1 "No-futz" commodity storage

Grid storage does have some of the same storage requirements as other systems. Ease of integration and maintenance is one of these requirements. Grid users, like all users, need appliance-like storage that allows true "no-futz" computing. Many of the current vendor-offered storage appliances are easily installed and maintained.

However, these appliances bundle together proprietary hardware and software and are sold at prices well above the commodity curve. Although all users would prefer cheaper appliances, grid systems in particular need the ability to turn commodity machines into storage appliances on demand. Software based storage appliances can return storage to the commodity curve and enable grid systems to quickly react to storage needs through the immediate acquisition of storage appliances.

2.2 Multiple administrative domains

One of the more unique features of grid computing is that grid users migrate across multiple administrative domains. This migration presents three rather unique challenges to grid storage. The first is that each of these administrative domains may use a different set of communication and security protocols. This migration creates an additional challenge in that grid storage must provide efficient methods of access for both local and wide area storage transfers. Finally, providing storage across administrative domains requires deploying storage appliances within each domain. System administrators are justifiably wary of allowing arbitrary code super-user privilege within their domains. Any proposed storage appliance must therefore be able to run without special privilege in order to be politically acceptable on the grid.

2.3 User Management and Reservations

A storage appliance must manage both its users and their groups. The grid introduces a new type of migratory users, who use storage as a hopping board rather as a long-term store. Support for migratory users and their storage demands should be efficiently provided by the storage appliance.

Reservation guarantees and different qualities of service are storage requirements that are perhaps felt more

[1] Although computational storage is generally thought of as disks, it can be more generally defined as any physical capacity for storing and retrieving data.

strongly on the grid than elsewhere. Users will be unwilling to migrate to a different domain unless the storage appliance in that domain can guarantee to support the user's I/O demands. Storage appliances must therefore provide mechanisms by which users can reserve storage space for their I/O requirements. Beyond just reservations, the storage appliance should also be "cost-aware", and able to differentiate beyond valuable and worthless data when expired reservations are selected for removal.

2.4 Policy language support

For a storage appliance to be appropriate for grid computing, it must provide flexible mechanisms to advertise its capacity, its availability and its contents. Administrators will want additional mechanisms to allow them to specify who will use their storage appliances, how much storage users will be allowed and for how long will they be allowed to use it.

The Condor scheduling system [8] has the ability to matchmake [10] resources and consumers using administrative policies defined in the ClassAd policy language. Condor's wide-spread deployment on the grid stands as a testimony to the necessity for flexibly policy language support in storage appliances.

3 A unique solution: NeST

NeST is a user-level software-only storage appliance that is being specifically designed to meet the unique storage needs of grid computation. A virtual protocol layer allows arbitrary data transfer and authentication protocols to be used. An abstract storage interface allows a wide range of physical storage devices to be optimally utilized. Multiple concurrency architectures provide the flex needed to achieve high levels of throughput across multiple host platforms. Flexible mechanisms allow storage within NeST to be reserved and there is support for differentiated levels of service as well. Finally, a policy language is used to advertise the NeST and allow discovery of both the physical storage and of the data contained thereon.

3.1 From commodity to appliance

To break the false dependence imposed by current vendors between hardware and software, NeST is a software only storage appliance. The advantage of this is twofold: cost and convenience. By decoupling the appliance from the hardware, NeST is able to return the cost of storage to the commodity curve. In addition to being inexpensive, commodity machines are typically easily available, thereby making the acquistion of a NeST storage appliance particularly easy. From the start, our goal has been the "no-futz" conversion of a commodity machine into a storage appliance.

However, attempting to provide appliance-like behavior across a wide range of commodity systems (both hardware and software) does create new challenges. The appliance must be able to adapt to the particulars of the host commodity machine. Storage appliances such as those provided by NetApp and EMC avoid these challenges by building integrated appliances in which the storage must interface with only one operating system and one physical storage interface. To meet these challenges and realize our goal of allowing users to create appliances from commodities, NeST has an abstract storage interface and uses multiple concurrency architectures.

3.1.1 Abstract storage interface

There is a wide range of physical storage devices found in current commodity systems. For example, a machine might have a hardware RAID system, IDE or SCSI disks, or perhaps even a tape drive. In addition, the file systems provided by different commodity operating systems may have radically different performance on the same hardware. For this reason, NeST is designed such that all interaction with the local filesystem is directed through an abstract storage interface.

This allows NeST to transparently adapt to the particulars of the physical storage and file system of the host machine. For example, NeST should use the local file system when run on operating systems with a good local file system and should manage directly the raw disk when run on operating systems without. The abstract storage interface can allow this by providing multiple virtual storage implementations and intelligently, perhaps even dynamically, choosing between them.

This abstract storage layer would even permit a storage appliance to be created from a diskless workstation. This memory only storage appliance would expose a filesystem interface to remote memory. This could then be used in a variety of ways. One example is as high speed buffers by wide area data movement systems such as Kangaroo. [13]

3.1.2 Multiple concurrency architectures

Another difficulty of creating storage appliances from commodity machines is that models to achieve high concurrency are not consistent across different operating systems. [9] For example, using threads to multiplex between multiple clients may work well on some systems but poorly on others.

To address this, NeST is currently implemented with three different, swappable, concurrency architectues.

These is an architecture which dispatches client requests to a pre-allocated pool of processes, an architecture which dispatches client requests to a pre-allocated pool of threads and a single process non-blocking architecture based on the `select` system call. For each host system, NeST can therefore use the architecture with the highest level of performance on that particular system.

3.2 Virtual protocol layer

To meet the challenges of providing storage for multiple administrative domains, NeST exports a virtual protocol layer. Multiple administrative domains will use different protocols for wide area file transfers, for local area accesses and will often need to use their own proprietary protocols as well. In addition to using different protocols for file transfers, various adminstrative domains will often require different protocols for security and authentication as well. For example, grid users who need to transfer data between two administrative domains require storage appliances that support all protocols necessary to authenticate and communicate with both domains.

However, to build a storage appliance that supports all necessary protocols for all administrative domains *ipso facto* is unrealistic. On the other hand, a storage appliance that is *capable* of using arbitrary protocols is both feasible and necessary for grid computing.

A virtual protocol layer allows NeST to communicate using multiple protocols simultaneously and new protocols are easily added in the same way that new file systems can be added in a virtual file system (VFS) [7]. Similarly, multiple authentication mechanisms can be used by adding them to the connection routines of the virtual protocols. The VFS analogy is particularly relevant because, by adding a virtual protocol to NeST, the effect is that of creating a pluggable file system on the storage appliance. For example, to efficiently allow both wide and local storage accesses, NeST could be configured with a virtual grid-FTP protocol for the wide area accesses and a virtual NFS protocol for the local.

3.2.1 Chirp

The difficulty that we anticipated in creating a virtual protocol layer was defining the minimum subset of generic storage requests such that any and all network protocols can cleanly mesh with the interface. Although we have implemented only a small number of protocols to date, we have found that the minimum subset defined so far is somewhat lacking. Common to the protocols which we have examined are the obvious file access requests such as get, put, list and remove. What is lacking however are the meta-management requests of the storage appliance itself, such as requests to add users, set caching policy, request statistics and more flexible mechanisms to allow the expression of administrative policies as well.

For this reason, we have begun designing a new network protocol, Chirp, that contains the minimum comprehensive set of file access requests as listed above as well as the meta-management requests. Clients who are unable to rebuild their applications to take advantage of the specific Chirp commands can still of course use the protocol of their choice. However, clients who can rebuild their applications can use the managerial features in Chirp to attempt to optimize their use of the appliance.

Our expectation is that scheduling systems like Condor will be the primary users of the Chirp protocol. For example, using Chirp, Condor could create an account for a grid user in a local Condor pool, create a reservation for the user's output files, ensure that the input files have been staged and then schedule the job. The job meanwhile does not need to be Chirp aware, or even NeST aware, and can transparently access its data through an interposition agent like Bypass. [14]

3.3 User Management and Reservations

As mentioned in section 2.3, users will only be willing to migrate data files across wide areas if they can be assured *a priori* of reliable storage allocation. For this reason, we have defined and added the necessary set of Chirp requests to the abstract storage interface.

3.3.1 Reservations and guarantees

A storage appliance designed to be shared by multiple users must allow the storage space available to a user to be restricted. NeST allows this by accepting storage space reservation requests. Currently, only the Chirp protocol is capable of sending these requests, but any protocol which supports reservations could plug into the virtual protocol layer and take advantage of this functionality. All storage space reservation requests in NeST consist of three parameters - storage *size*, the *time* duration for which the reservation is required and the reproduction-*cost* estimate of the data to be stored in the reservation. The time and cost parameters in reservation requests can be correctly manipulated to accommodate migratory users.

Different levels of service are provided via the cost parameter associated with each reservation. Reservations (and their associated data) are removed lazily when a new reservation request can be fulfilled only by removing already expired reservations. The victim selection process is a function of the cost parameter and the time since expiration.

The reservation system is intricately linked with the storage system and as such is implemented within the abstract storage layer. Currently the abstract storage layer has been defined but not yet well fleshed out and contains only one implementation, that which uses the local filesystem. In this case, the native quota system provided by the filesystem seems to be the natural choice for enforcing these reservations. The advantages in using the filesystem's quota management are introducing very little overhead on system's performance, its ability to operate on the granularity of blocks/inodes per device, and does not require any kernel changes for using it.

On the other hand, the filesystem's quotas can be manipulated only by the super-user. When run without super-user privilege, NeST cannot protect the storage device from other users and therefore must treats reservation requests as hints rather than guarantees. Additionally, the filesystem provided quota system is not device-size aware; in other words, the quota system can not by itself guarentee that the sum of all quotas given to users in a particular device is less than or equal to the size of the device. Therefore, it is the responsibility of the NeST to ensure that storage is not overbooked.

We decided to use the quota system as the enforcement mechanism as we could not find any other effective mechanism that satisfies our needs. Logical Volume Manager (LVM) [11] could be used as the enforcement mechanism, but the operating costs involved are very high. The cost of creating a new logical volume for every new user and the cost of changing the size of a logical volume every time the user's reservation needs to be updated are very high compared to the filesystem's quota updates. LVM also allows only the super-user to change or reconfigure the logical volumes.

3.3.2 User and group management

In addition to reservation requests, NeST provides a set of requests to manage users and groups. These requests are analogous with similar requests in the Unix operating system. Their implementation within NeST is dependent on the abstract storage interface as well as whether NeST is run with super-user privilege. In the filesystem implementation, NeST can use the system's user *(/etc/passwd)* and group *(/etc/group)* management files to manage its users/groups. When run without super-user privileges, NeST can manage its own user and group files. In this case, all files appear to the operating system to be actually owned by NeST.

3.4 Policy language support

It is clear that policy languages are becoming an integral feature of the grid. Languages such as ClassAds [10] allow the expression of arbitrary policies. Given the difficulties created by multiple administrative domains, flexible policy languages are necessary to allow system administrators the freedom to deploy and use storage appliances within the constraints of their own particular policies.

NeST's virtual protocol layer allows arbitrary policy languages to act as an interface or a broker to the storage appliance. Such flexible mechanisms make it difficult to predict the range of policies which might be applied to a storage appliance. As a exploratory step in this direction, we are adding a few simple policies to Chirp using ClassAds and the Condor matchmaking system.

3.4.1 Administrative management

An important storage feature is the ability needed by administrators to set and enforce the policies of their administrative domain. Examples of these administrative decrees are time leased allocations, economic based allocations and mechanisms to define both how and to whom the resource is advertised.

These advertisements can then be used to guide the resource selection of both individual users and of scheduling systems. For example, the Chirp protocol currently allows users to name logical collections of data stored on a NeST. The NeST then advertises the logical collections it contains to the Condor matchmaker, which can then place jobs needing those collections in close proximity to the NeST. Other information is also currently included in these advertisements such as the total available space on the NeST as well as a list of supported protocols and the ports on which they are being monitored.

4 A case study

To demonstrate the utility of using NeST for grid computations, we have, jointly with other researchers here at UW, explored the creation of grid communities. [12] The motivation for this work is the large datasets accessed by many high-energy physics applications. [1] In this work, we propose creating grid communities around replicated datasets. Previous work had emphasized either moving jobs to the data or moving the data to the job.

Clearly both approaches have their limitations. By using the scheduling and matchmaking features of Condor, the transparent interpositioning of Bypass and storage provided by NeST, we were able to combine these approaches. Using the high-throughput provided by the GridFTP [2] virtual protocol, we first replicated the data to remotely distributed NeSTs. Around each NeST, an I/O community was thereby formed as shown in Figure 1. Scheduled jobs then used Bypass and Chirp to perform partial file accesses. This approach combines the benefits of both job moving and data moving and allows scaling,

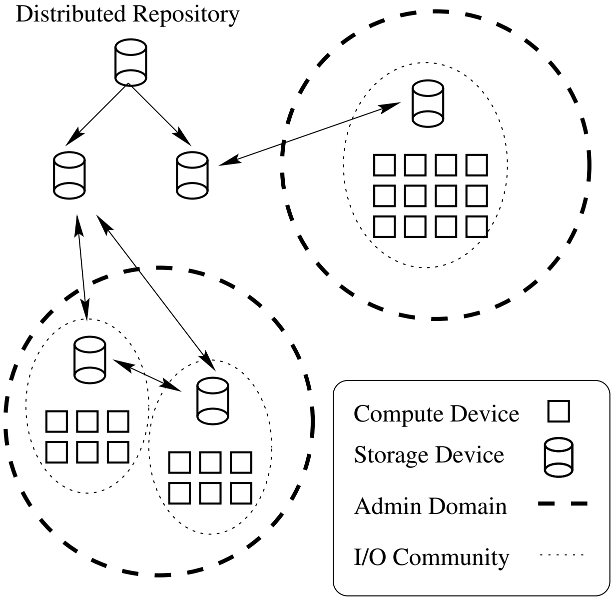

Figure 1: **I/O Communities.** *A dataset is replicated from the distributed repository and staged in remotely distributed NeSTs. Around each NeST, an I/O community is formed and Condor can then schedule jobs within the newly created community. Bypass allows the scheduled jobs to transparently use Chirp to access the data from the NeST. This approach allows scaling, sharing and low-latency data accesses.*

sharing and low-latency data accesses. Experimental results using remotely distributed NeSTs have not yet been collected, but local experiments suggest that the expected speedup is significant.

5 Future work

This is a wide-open area. Clearly there is a storage void currently in the grid. NeST is positioned to fill that void. As we've argued, NeST has been designed for the grid and is the only known storage appliance that can satisfy the diverse and challenging set of requirements created by grid computing.

But much work needs to be done before NeST is a production quality system. Better scheduling and control of multiple concurrent transfers is one area which needs to be examined. Grid schedulers need storage which can provide at least approximate schedule guarantees. To enable such scheduling, we will need to add bandwidth reservation to our current reservation mechanism which allows storage reservations only.

Storage bandwidth reservations would have two components, the network bandwidth and the disk bandwidth; the bandwidth realized by the user would be the minimum of the two. Allocating desired network bandwidth to multiple users is not overly difficult and could be implemented using prioritized time division multiplexing. Disk bandwidth allocations however are more involved as they involve an additional seek time. Mere multiplexing of disk accesses could result in multiple seeks and reduce available throughput. Additionally, scheduling disk accesses is not straightforward as the device controller may reorder disk access requests.

The virtual protocol layer could use some more prodding to attempt to better define the comprehensive subset of storage requests necessary to allow arbitrary communication protocols to be easily added. The abstract storage interface is designed but needs to be fleshed out. Currently it supports only one storage implementation: a flat namespace on the local file system. Flexible mechanisms for guaranteeing different levels of storage bandwidth need to be designed and implemented.

Finally, NeST should eventually be able to automatically configure its own concurrency architecture and storage interface either at installation time or dynamically.

6 Conclusion

The grid presents many new and as of yet unmet challenges to storage appliances. The NeST storage appliance recognizes the unique nature of grid computing and is designed specifically for the grid.

By building software-only appliances, NeST returns storage to the commodity curve and facilitates on demand creation of storage appliances. An abstract storage interface and multiple concurrency architectures are designed to provide consistent levels of performance across multiple commodity systems. Flexible mechanisms to reserve storage space and provide different levels of service are present as well.

Grid applications especially can benefit from NeST's virtual protocol layer which allows arbitrary communication and authentication protocols to access the appliance. This virtual protocol layer also allows administrators to express and enforce adminsistrative policies.

It has also been shown that NeST can be effectively manipulated by scheduling systems to allow efficient resource usage across widely distributed grids. In this way, NeST storage appliances can be both easily deployed and easily discovered.

References

[1] William Allcock, Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steve Tuecke.

The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. to appear in the *Journal of Network and Computer Applications*, 2001.

[2] Ann Chervenak, Ian Foster, Carl Kesselman, and Steve Tuecke. Protocols and services for distributed data-intensive science. to appear in *Proceedings of ACAT2000*, 2000.

[3] EMC Corporation. http://www.emc.com.

[4] Ian Foster, Carl Kesselman, and Steve Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. to appear in *International Journal of Supercomputer Applications*, 2001.

[5] D. Hitz. A storage networking appliance. Technical Report TR3001, Network Appliance, Inc., 2000.

[6] Network Appliance Inc. http://www.netapp.com.

[7] S. R. Kleiman. Vnodes: An architecture for multiple file system types in Sun Unix. In *Proc. Summer 1986 USENIX Conf.*, pages 238–247, Atlanta, GA (USA), 1986.

[8] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - A hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS)*, pages 104–111, Washington, DC, 1988. IEEE Computer Society.

[9] V. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *Proceedings of the Usenix Technical Conference*, 1999.

[10] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 1998.

[11] David Teigland and Heinz Mauelshagen. Volume managers in linux. In *Proceedings of the 2001 USENIX Annual TechnicalConference*, 2001.

[12] Doug Thain, John Bent, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, and Miron Livny. Gathering at the well: Creating communities for grid i/o. submitted to SuperComputing, 2001.

[13] Douglas Thain, Jim Basney, Se-Chang Son, and Miron Livny. The kangaroo approach to data movement on the grid. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing*, San Francisco, California, August 2001.

[14] Douglas Thain and Miron Livny. Multiple bypass: Interposition agents for distributed computing. *Journal of Cluster Computing*, 4:39–47, 2001.