Chapter 1

NEST - A GRID ENABLED STORAGE APPLIANCE

John Bent, Venkateshwaran Venkataramani,
Nick LeRoy, Alain Roy, Joseph Stanley,
Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau,
and Miron Livny

Abstract We describe NeST, a flexible software-only storage appliance designed to meet the storage needs of the Grid. NeST has three key features that make it well-suited for deployment in a Grid environment. First, NeST provides a generic data transfer architecture that supports multiple data transfer protocols (including GridFTP and NFS), and allows for the easy addition of new protocols. Second, NeST is dynamic, adapting itself on-the-fly so that it runs effectively on a wide range of hardware and software platforms. Third, NeST is Grid-aware, implying that features that are necessary for integration into the Grid, such as storage space guarantees, mechanisms for resource and data discovery, user authentication, and quality of service, are a part of the NeST infrastructure. We include a practical discussion about building grid tools using the NeST software.

Keywords: Storage, storage management, storage appliances, reservations, quality of service, data management, Grid, Condor, scheduling, data transfer, client-server.

1. INTRODUCTION

Data storage and movement are of increasing importance to the Grid. Over time, scientific applications have evolved to process larger volumes of data, and thus their overall throughput is inextricably tied to the timely delivery of data. As the usage of the Grid evolves to include commercial applications (Lohr, 2002), data management will likely become even more central than it is today.

Data management has many aspects. While performance has long been the focus of storage systems research, recent trends indicate that other factors, including reliability, availability, and manageability, may now be more relevant.

In particular, many would argue that manageability has become the dominant criterion in evaluating storage solutions, as the cost of storage management outweighs the cost of the storage devices themselves by a factor of three to eight (Patterson, 2002).

One potential solution to the storage management problem is the use of specialized storage devices known as *appliances*. Pioneering products such as the filers of Network Appliance (Hitz et al., 1994) reduce the burden of management through *specialization*; specifically, their storage appliances are designed solely to serve files to clients, just as a toaster is designed solely to toast. The results are convincing: in field testing, Network Appliance filers have been shown to be easier to manage than traditional systems, reducing both operator error and increasing system uptime considerably (Lancaster and Rowe, 2001).

Thus, storage appliances seem to be a natural match for the storage needs of the Grid, since they are easy to manage and provide high performance. However, there are a number of obstacles that prevent direct application of these commercial filers to the Grid environment. First, commercial storage appliances are inflexible in the protocols they support, usually defaulting to those common in local area Unix and Windows environments (*e.g.*, NFS (Walsh et al., 1985) and CIFS (Sharpe, 1999)). Therefore, filers do not readily mix into a world-wide shared distributed computing infrastructure, where non-standard or specialized Grid protocols may be used for data transfer. Second, commercial filers are expensive, increasing the cost over the raw cost of the disks by a factor of ten or greater. Third, storage appliances may be missing features that are crucial for integration into the Grid environment, such as the ability to interact with larger-scale global scheduling and resource management tools.

To overcome these problems and bring appliance technology to the Grid, we introduce NeST, an open-source, user-level, software-only storage appliance. As compared to current commercial storage appliances, NeST has three primary advantages: flexibility, cost, and Grid-aware functionality. We briefly discuss each of these advantages in more detail.

First, NeST is more flexible than commercial storage appliances. NeST provides a generic data transfer architecture that concurrently supports multiple data transfer protocols (including GridFTP (Allcock et al., 2002) and NFS). The NeST framework also allows new protocols to be added as the Grid evolves.

Second, because NeST is an open-source software-only appliance, it provides a low-cost alternative to commercial storage appliances; the only expenses incurred are the raw hardware costs for a PC with a few disks. However, because NeST is a software-based appliance, it introduces new problems that traditional appliances do not encounter: NeST must often run on hardware that it was not tailored for or tested upon. Therefore, NeST contains the ability

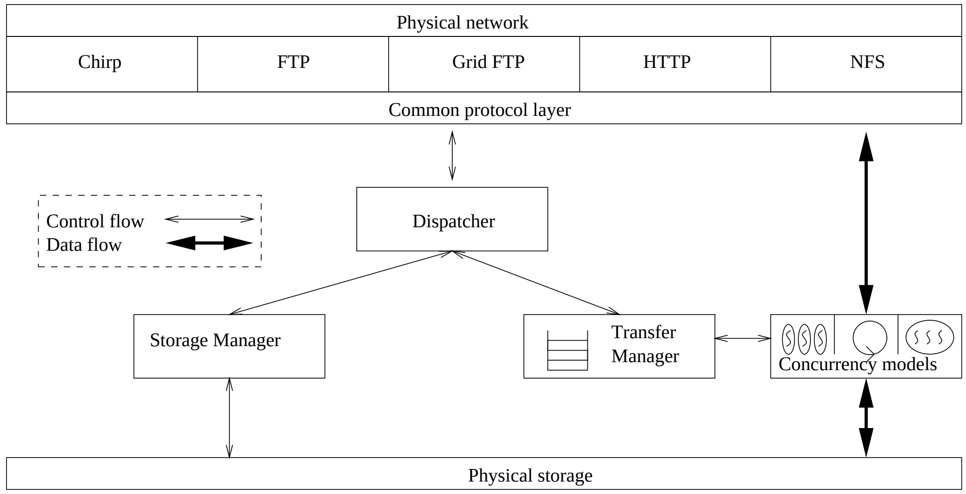

Figure 1.1. **NeST Software Design.** *The diagram depicts NeST and its four major components: the protocol layer, the storage manager, the transfer manager, and the dispatcher. Both control and data flow paths are depicted.*

to adapt to the characteristics of the underlying hardware and operating system, allowing NeST to deliver high performance while retaining the ease of management benefits of storage appliances.

Finally, NeST is Grid-aware. Key features, such as storage space guarantees, mechanisms for resource and data discovery, user authentication, and quality of service, are a fundamental part of the NeST infrastructure. This functionality enables NeST to be integrated smoothly into higher-level job schedulers and distributed computing systems (**?**; **?**; **?**; **?**; **?**; **?**).

The rest of this chapter is organized as follows. Section 2 describes the overall design of NeST. The protocol layer which mediates interaction with clients is described in Section 3. Section 4 describes the transfer manager which is responsible for monitoring and scheduling concurrency and quality of service, and Section 5 describes the storage layer which manages the actual physical storage of the system. An example usage of NeST is traced within Section 6, Section 7 describes the user interface, comparisons to related work are in Section 8, and conclusions are drawn in Section 9.

2. DESIGN OVERVIEW

As a Grid storage appliance, NeST provides mechanisms both for file and directory operations as well as for resource management. The implementation to provide these mechanisms is heavily dependent upon NeST's modular design, shown in Figure 1.1. The four major components of NeST are its *protocol layer, storage manager, transfer manager* and *dispatcher*. We first briefly examine each of these components separately; then we show how they work together by tracing an example client interaction.

2.1 Component Descriptions

The *protocol layer* in NeST provides connectivity to the network and all client interactions are mediated through it. Clients are able to communicate with NeST with any of the supported file transfer protocols, including HTTP (Fielding et al., 1997), NFS (Sandberg, 1985), FTP (Postel, 1980), GridFTP (Allcock et al., 2002), and Chirp, the native protocol of NeST. The role of the protocol layer is to transform the specific protocol used by the client to and from a common request interface understood by the other components in NeST. We refer to this as a *virtual protocol connection* and describe it and the motivation for multiple protocol support in Section 3.

The *dispatcher* is the main scheduler and macro-request router in the system and is responsible for controlling the flow of information between the other components. It examines each client request received by the protocol layer and routes each appropriately to either the storage or the transfer manager. Data movement requests are sent to the transfer manager; all other requests such as resource management and directory operation requests are handled by the storage manager. The dispatcher also periodically consolidates information about resource and data availability in the NeST and can publish this information as a ClassAd (**?**) into a global scheduling system (Thain et al., 2001b).

The *storage manager* has four main responsibilities: virtualizing and controlling the physical storage of the machine (*e.g.*, the underlying local filesystem, raw disk, physical memory, or another storage system), directly executing non-transfer requests, implementing and enforcing access control, and managing guaranteed storage space in the form of *lots*. Lots are discussed in more detail below.

Because these storage operations execute quickly (in the order of milliseconds), we have chosen to simplify the design of the storage manager and have these requests execute synchronously. It is the responsibility of the dispatcher to ensure that storage requests are serialized and executed at the storage manager in a thread-safe schedule.

The *transfer manager* controls data flow within NeST; specifically, it transfers data between different protocol connections (allowing transparent three- and four-party transfers). All file data transfer operations are managed asynchronously by the transfer manager after they have been synchronously approved by the storage manager. The transfer manager contains three different concurrency models, threads, processes and events, and schedules each transfer using one of these models. Scheduling policies, such as preferential scheduling, and scheduling optimizations are the responsibility of the transfer manager and are discussed in Section 4.

2.2 An Example Client Interaction

We now examine how these four components function together by tracing the sequence of events when interacting with a client. In this example, we consider the case when a client first creates a new directory (*i.e.*, a non-transfer request) and then inserts a file into that directory (*i.e.*, a transfer request).

When the client initially connects to NeST with the request to create the directory, the dispatcher wakes and asks the protocol layer to receive the connection. Depending upon the connecting port, the protocol layer invokes the handler for the appropriate protocol. The handler then authenticates the client, parses the incoming request into the common request format, and returns a virtual protocol connection to the dispatcher.

The dispatcher then asks the storage manager to create the directory. After checking for access permissions, the storage manager synchronously creates the directory and sends acknowledgment back to the client through the dispatcher and the virtual protocol connection.

At this point, the dispatcher assumes responsibility for the client and listens for further requests on its channel. After the client sees that the directory is created successfully, it requests permission to send a file. The dispatcher invokes its virtual protocol connection to receive this request and again queries the storage manager. The storage manager allows the transfer and returns a virtual protocol connection into which the transfer can be written. The dispatcher passes both connections to the transfer manager, stops listening on the client channel, and sleeps, waiting for the next client request.

The transfer manager is then free to either schedule or queue the request; once the request is scheduled, the transfer manager uses past information to predict which concurrency model will provide the best service and passes the connection to the selected model. The transfer continues as the chosen concurrency model transfers data from the client connection to the storage connection, performing an acknowledgment to the client if desired. Finally, the transfer status is returned to the transfer manager and then up to the dispatcher.

In the following sections, we describe the most important aspects of NeST. First, we motivate the importance of supporting multiple communication protocols within a virtual protocol layer. Second, we describe how the transfer manager adapts to the client workload and underlying system to pick the concurrency model with the best performance. Third, we show how the transfer manager can apply scheduling policies among different connections. Fourth, we explain the role of storage guarantees in NeST, and explain how the storage manager implements this functionality.

3. PROTOCOL LAYER

Supporting multiple protocols is a fundamental requirement of storage appliances used in the Grid. Though there has been some standardization toward a few common protocols within the Global Grid Forum (gri,), diversity is likely to reign in a community as widespread and fast-moving as the Grid. For example, even if all wide-area transfers are conducted via GridFTP, local-area file access will still likely be dominated by NFS, AFS, and CIFS protocols.

Multiple protocols are supported in NeST with a virtual protocol layer. The design and implementation of our virtual protocol layer not only allows clients to communicate with NeST using their preferred file transfer protocol, but also shields the other components of NeST from the detail of each protocol, allowing the bulk of NeST code to be shared among many protocols. Thus, the virtual protocol layer in NeST is much like the the virtual file system (VFS) layer in many operating systems (Kleiman, 1986).

An alternative approach to having a single NeST server with a virtual protocol layer is to implement separate servers that understand each individual protocol and run them simultaneously; we refer to this latter approach as "Just a Bunch Of Servers" or "JBOS". The relative advantage of JBOS is that servers can be added or upgraded easily and immediately once any implementation of that protocol is available; with NeST, incorporating a new or upgraded protocol may take more effort, as the protocol operations must be mapped onto the NeST common framework.

However, we believe the advantages of a single server outweigh this implementation penalty for a number of reasons. First, a single server enables complete control over the underlying system; for example, the server can give preferential service to requests from different protocols or even to different users across protocols. Second, with a single interface, the tasks of administering and configuring the NeST are simplified, in line with the storage appliance philosophy. Third, with a single server, optimizations in one part of the system (*e.g.*, the transfer manager or concurrency model) are applied to all protocols. Fourth, with a single server, the memory footprint may be considerably smaller. Finally, the implementation penalty may be reduced when the protocol implementation within NeST can leverage existing implementations; for example, to implement GridFTP, we use the server-side libraries provided in the Globus Toolkit and we use the Sun RPC package to implement the RPC communication in NFS.

At this point, we have implemented five different file transfer protocols in NeST: HTTP, NFS, FTP, GridFTP, and the NeST native protocol, Chirp. In our experience, most request types across protocols are very similar (*e.g.*, all have directory operations such as `create`, `remove`, and `read`, as well as file operations such as `read`, `write`, `get`, `put`, `remove`, and `query`) and fit easily into our virtual protocol abstraction. However, there are interesting exceptions; for instance, Chirp is the only protocol that supports lot management (lots will be discussed further in Section 5) and NFS is the only protocol with a lookup and mount request. Note that mount, not technically part of NFS, is actually a protocol in its own right; however, within NeST, mount is handled by the NFS handler.

We plan to include other Grid-relevant protocols in NeST, including data movement protocols such as IBP (Plank et al., 2001) and resource reservation protocols, such as those being developed as part of the Global Grid Forum (gri,). We expect that as new protocols are added, most implementation effort will be focused on mapping the specifics of the protocol to the common request object format, but that some protocols may require additions to the common internal interface.

Since the authentication mechanism is protocol specific, each protocol handler performs its own authentication of clients. The drawback of this approach is that a devious protocol handler can falsify whether a client was authenticated. Currently, we allow only Grid Security Infrastructure (GSI) authentication (Foster et al., 1998), which is used by Chirp and GridFTP; connections through the other protocols are allowed only anonymous access.

4. TRANSFER MANAGER

At the heart of data flow within NeST is the transfer manager. The transfer manager is responsible for moving data between disk and network for a given request. The transfer manager is *protocol agnostic*: thus, all of the machinery developed within the manager is generic and moves data for all of the protocols, highlighting one of the advantages of the NeST design.

4.1 Multiple Concurrency Models

Inclusion in a Grid environment mandates the support for multiple on-going requests. Thus, NeST must provide a means for supporting concurrent transfers. Unfortunately, there is no single standard for concurrency across operating systems: on some platforms, the best choice is to use threads, on others, processes, and in other cases, events. Making the decision more difficult is the fact that the choice may vary depending on workload, as requests that hit in the cache may perform best with events, and those that that are I/O bound perform best with threads or processes (Pai et al., 1999).

To avoid leaving such a decision to an administrator, and to avoid choosing a single alternative that may perform poorly under certain workloads, NeST implements a flexible *concurrency architecture*. NeST currently supports three models of concurrency (threads, processes, and events), but in the future we plan to investigate more advanced concurrency architectures (*e.g.*, SEDA (Welsh et al., 2001) and Crovella's experimental server (Crovella et al., 1999)). To deliver high performance, NeST dynamically chooses among these architectures; the choice is enabled by distributing requests among the architectures equally at first, monitoring their progress, and then slowly biasing requests toward the most effective choice.

4.2 Scheduling

Because there are likely to be multiple outstanding requests within a NeST, NeST is able to selectively reorder requests to implement different scheduling policies. When scheduling multiple concurrent transfers, a server must decide how much of its available resources to dedicate to each request. The most basic strategy is to service requests in a first-come, first-served (FCFS) manner, which NeST can be configured to employ. However, because the transfer manager has control over all on-going requests, many other scheduling policies are possible. Currently, NeST supports both *proportional share* and *cache-aware* scheduling in addition to FCFS.

4.2.1 Quality of service

Proportional-share scheduling (Waldspurger and Weihl, 1995) is a deterministic algorithm that allows fine-grained proportional resource allocation and has been used previously for CPU scheduling and in network routers (Kohler et al., 2000). Within the current implementation of NeST, it is used to allow the administrator to specify proportional preferences per protocol class (*e.g.*, NFS requests should be given twice as much bandwidth as GridFTP requests); in the future, we plan to extend this to provide preferences on a per-user basis.

Using byte-based strides, this scheduling policy accounts for the fact that different requests transfer different amounts of data. For example, an NFS client who reads a large file in its entirety issues multiple requests while an HTTP client reading the same file issues only one. Therefore, to give equal bandwidth to NFS requests and HTTP requests, the transfer manager schedules NFS requests N times more frequently, where N is the ratio between the average file size and the NFS block size.

NeST proportional share scheduling is similar to the Bandwidth and Request Throttling module (Howe, 2000) available for Apache. However, proportional share scheduling in NeST offers more flexibility because it can schedule across multiple protocols, whereas Apache request-throttling only applies to the HTTP requests the Apache server processes, and thus cannot be applied to other traffic streams in a JBOS environment.

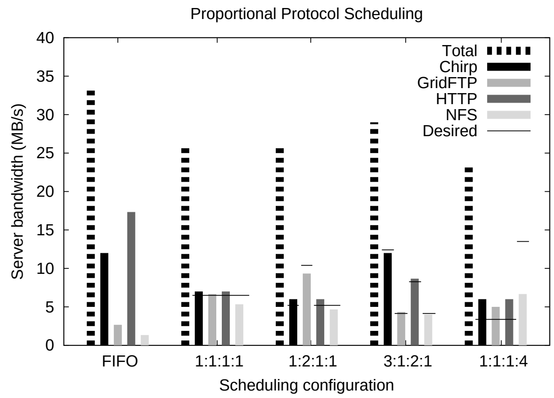

Figure 1.2. **Proportional Protocol Scheduling.** *This graph measures the fairness and overhead of quality of service scheduling in a NeST running a synthetic workload. Within each set of bars, the first bar represents the total delivered bandwidth across all protocols; the remaining bars show the bandwidth per protocol. The labels for the sets of bars show the specified proportional ratios; the desired lines show what the ideal proportions would be. Note that NeST is able to achieve very close to the desired ratios in each case except the right-most.*

The overhead and achieved fairness of proportional share scheduling in NeST is shown in Figure 1.2. The first set of bars shows our base case in which the NeST transfer manager uses the simple FIFO scheduler. The other sets of bars adjust the desired ratio of bandwidth for each protocol.

We can make two conclusions from this graph. First, the proportional share scheduler imposes a slight performance penalty over FIFO scheduling, delivering a total of approximately 24-28 MB/s instead of 33 MB/s. Second, the proportional-share scheduler achieves very close to the desired ratios in almost all cases. Specifically, using Jain's metric (Chiu and Jain, 1989) of fairness in which a value of 1 represents an ideal allocation, we achieve values of greater than 0.98 for the 1:1:1:1, the 1:2:1:1, and the 3:1:2:1 ratios.

The only exception is that allocating additional bandwidth to NFS (*e.g.*, 1:1:1:4 for Chirp:GridFTP:HTTP:NFS) is extremely difficult; the Jain's fairness value in this case drops to 0.87. The challenge is that due to the smaller block size used by NFS there are not a sufficient number of NFS requests for the transfer manager to schedule them at the appropriate interval; in the case where there is no available NFS request, our current implementation is work-conserving and schedules a competing request, rather than allow the server to be idle. We are currently implementing a non-work-conserving policy in which the idle server waits some period of time before scheduling a competitor (Iyer and Druschel, 2001); such a policy might pay a slight penalty in average response time for improved allocation control.

4.2.2 Cache-aware scheduling

Cache-aware scheduling is utilized in NeST to improve both average client perceived response time as well as server throughput. By modeling the kernel buffer cache using gray-box techniques (Arpaci-Dusseau and Arpaci-Dusseau, 2001), NeST is able to predict which requested files are likely to be cache resident and can schedule them before requests for files which will need to be fetched from secondary storage. In addition to improving client response time by approximating shortest-job first scheduling, this scheduling policy improves server throughput by reducing the contention for secondary storage.

In earlier work (Burnett et al., 2002), we examined cache-aware scheduling with a focus toward web workloads; however, given the independence between the transfer manager and the virtual protocol layer, it is clear that this policy works across all protocols. This illustrates a major advantage that NeST has over JBOS in that optimizations in the transfer code are immediately realized across all protocols and need not be reimplemented in multiple servers.

5. STORAGE MANAGER

Much as the protocol layer allows multiple different types of network connections to be channeled into a single flow, the storage manager has been designed to virtualize different types of physical storage and to provide enhanced functionality to properly integrate into a Grid environment. The three specific roles fulfilled by the storage manager are to implement access control, virtualize the storage namespace, and to provide mechanisms for guaranteeing storage space.

Access control is provided within NeST via a generic framework built on top of collections of ClassAds (**?**). AFS-style access control lists determine read, write, modify, insert, and other privileges, and the typical notions of users and groups are maintained. NeST support for access control is generic, as these policies are enforced across any and all protocols that NeST supports; clients need only be able to communicate via the native Chirp protocol (or any supported protocol with access control semantics) to set them.

NeST also virtualizes the physical namespace of underlying storage, thus enabling NeST to run upon a wide variety of storage elements. However, in our current implementation, we currently use only the local filesystem as the underlying storage layer for NeST; we plan to consider other physical storage layers, such as raw disk, in the near future.

When running in a remote location in the Grid, higher-level scheduling systems, individual users and Grid middleware, such as SRM (**?**), all must be assured that there exists sufficient storage space to save the data produced by their computation, or to stage input data for subsequent access. To address this problem, NeST provides an interface to guarantee storage space, called a *lot*, and allows requests to be made for space allocations (similar to reservations for network bandwidth (Zhang et al., 1993)).

Each lot is defined by four characteristics: owner, capacity, duration, and files. The owner is the client entity allowed to use that lot; only individual owners are currently allowed but group lots will be included in the next release. The capacity indicates the total amount of data that can be stored in the lot. The duration indicates the amount of time for which this lot is guaranteed to exist. Finally, each lot contains a set of files; the number of files in a lot is not bounded and a file may span multiple lots if it cannot fit within a single one.

When the duration of a lot expires, the files contained in that lot are not immediately deleted. Rather, they are allowed to remain indefinitely until their space needs to be reclaimed to allow the creation of another new lot. We refer to this behavior as *best-effort* lots and are currently investigating different selection policies for reclaiming this space.

To create files on a NeST, a user must first have access to a lot; however, most file transfer protocols do not contain support for creating lots. In our environment, a lot can be obtained in two different ways. First, when system administrators grant access to a NeST, they can simultaneously make a set of default lots for users. Second, a client can directly use the Chirp protocol to create a lot before accessing the server with an alternative data-transfer protocol. Section 7 demonstrates via example how to use the Chirp protocol for lot managament.

To provide maximize flexibility and user-customization, NeST currently supports two different implementations of lots: kernel enforced and solely NeST-managed. Kernel enforced lots rely on the quota mechanism of the underlying filesystem, which allows NeST to limit the total amount of disk space allocated to each user. Utilizing the quota system affords a number of benefits: direct access to the file system (perhaps not through NeST) must also observe the quota restrictions, thus allowing clients to utilize NeST to make the space guarantee and then to bypass NeST and transfer data directly into a local file system. However, one limitation in this approach is that all of a user's lots must be consolidated into a single quota limit. This consolidation makes it therefore possible for a user to overfill one of her lots and then be unable to fill another of her lots to capacity. Another limitation of the kernel enforced lots is that the NeST must be run as super-user. The NeST-managed lot implementation does not suffer from these limitations but it does require that all access to a NeST machine be through one of the supported protocols. We leave it to the individual administrator to select which implementation is more appropriate for their users.

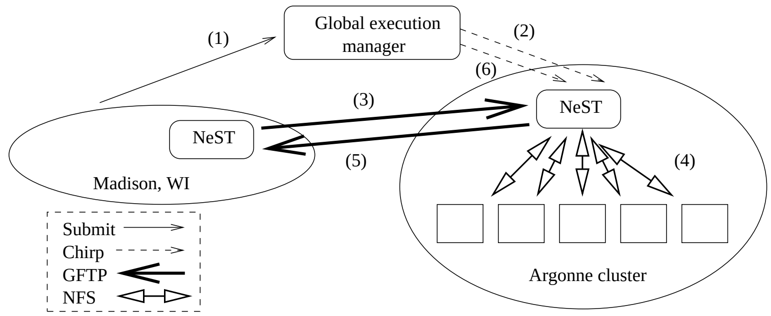

Figure 1.3. **NeST in the Grid.** *The diagram illustrates information flow in a scenario in which multiple NeST servers are utilized in the Grid.*

6. NeST IN THE GRID

With a basic understanding of NeST in place, we now illustrate how multiple NeST servers might be used in a global Grid environment. Figure 1.3 depicts such a scenario; all major events are labeled with the sequence numbers as defined in the following description.

In the figure, a user has their input data permanently stored at their *home site*, in this case at a NeST in Madison, Wisconsin. In step 1, the user submits a number of jobs for remote execution to a global execution manager. This manager is aware that a remote cluster, labeled the *Argonne cluster*, has a large number of cycles available. The NeST "gateway" appliance in the Argonne cluster has previously published both its resource and data availability into a global Grid discovery system (Thain et al., 2001b). The manager is therefore also aware that the Argonne NeST has a sufficient amount of available storage.

The manager decides to run the user's jobs at the Argonne site, but only after staging the user's input data there. Thus, in step 2, the manager uses Chirp to create a lot for the user's files at Argonne, thus guaranteeing sufficient space for input and output files. For step 3, the manager orchestrates a GridFTP third-party transfer between the Madison NeST and the NeST at the Argonne cluster. Other data movement protocols such as Kangaroo could also be utilized to move data from site to site (Thain et al., 2001a).

In step 4, the manager begins the execution of the jobs at Argonne, and those jobs access the user's input files on the NeST via a local file system protocol, in this case NFS. As the jobs execute, any output files they generate are also stored upon the NeST. Note that the ability to give preference to some users or protocols could be harnessed here, either by local administrators who wish to ensure preference for their jobs, or by the global manager to ensure timely data movement.

Finally, for step 5, the jobs begin to complete, at which point the manager moves the output data back to Madison, again utilizing GridFTP for the wide area movement. The manager is then free to use Chirp to terminate the lot in step 6, and inform the user that the output files are now available on the local NeST.

Note that many of the steps of guaranteeing space, moving input data, executing jobs, moving output data, and terminating reservations, can be encapsulated within a request execution manager such as the Condor Directed-Acyclic-Graph Manager (DAGMan) (Condor, 2002). Also, higher-level storage resource managers such as SRM could use NeST services to synchronize access between globally-shared storage resources (**?**).

7. USING NeST SOFTWARE

The NeSTsoftware is available for download here (Bent, 2003). Currently, NeST builds on both Linux and Solaris platforms. However, the kernel enforced lot implementation relies on Linux kernel utilities and is therefore not available with the Solaris version. Generally, using both the NeST client and server is comparable to using other file transfer software such as FTP. Therefore, most of this discussion will focus on using lots as they are a unique feature of NeST.

7.1 The NeST server

Using the NeST server is straightforward. Since NeST is user-level software that doesn't require any modified kernel patches, it can be installed and run simply. Please refer to the installation instructions on the webpage (Bent, 2003) for information about enabling the kernel quota system and downloading and installing necessary libraries such as the Globus Grid API bundle (prj, 2003). We recommend initially running NeST with the NeST-managed lot implementation instead of the kernel enforced version because it can be run without super-user privilege.

7.2 The NeST interactive client and client libraries

Because NeST supports many different protocols, clients can interact with the NeST server using any of the standard client programs such as GridFTP, FTP, HTTP and NFS. However, only the native Chirp protocol has lot management support. Included with the NeST software is a client library for putting Chirp protocol requests directly into user software and a thin interactive client program built from this library. In addition to all of the standard directory and data transfer operations, these client programs have support for user and group management as well as lot management. Figures 1.4 and 1.5 show how to use both the interactive client and the client library for lot management. Note that

```
chirp:db16:~johnbent/> lot user 10
    Lot Handle - 1
chirp:db16:~johnbent/> lot stat 1
    Lot Handle  : 1
    Lot Type    : Guaranteed
    OwnerUser   : johnbent
    Lot Size    : 10.000000 MB
    Lot Used    : 0.000000 MB (  0.00%)
    Start time  : 1/12/2003 12:34:00
    Duration    : 1440.00 Minutes
    Time left   : 1440.00 Minutes
chirp:db16:~johnbent/> lot update 1 0 10
chirp:db16:~johnbent/> put /usr/share/dict/linux.words
    409286 /     409286 (2.56 MB/s)
chirp:db16:~johnbent/> lot stat 1
    Lot Handle  : 1
    Lot Type    : Guaranteed
    OwnerUser   : johnbent
    Lot Size    : 10.000000 MB
    Lot Used    : 0.390326 MB (  3.90\%)
    Start time  : 1/12/2003 12:34:00
    Duration    : 1450.00 Minutes
    Time left   : 1448.72 Minutes
```

Figure 1.4. **Interactive nest-client.** *This shows one example server-client session using the provided interactive nest-client program. In this session, a user first creates a lot, then increases its duration by 10 minutes, writes a file into the lot and then queries the status of the lot.*

the Chirp protocol is used in both of these examples for both the lot management as well as the data transfer operations. However, in practice many users might prefer to use Chirp for lot management and then a different protocol such as GridFTP for the data transfers.

8. RELATED WORK

As a storage appliance, NeST relates most closely to the filers of Network Appliance (Hitz et al., 1994) and the Enterprise Storage Platforms of EMC (EMC Corporation, 2003). NeST does not attempt to compete with these commercial offerings in terms of raw performance as it is primarily intended for a different target domain. As such, NeST offers a low-cost, software-only alternative that offers more protocol flexibility and Grid-aware features that are needed to enable scientific computations in the Grid.

Within the Grid community, there are a number of projects that are related to NeST. GARA (**?**) is an architecture that provides advance reservations across a variety of resources, including computers, networks, and storage devices. Like

```
#include "nest_client.h"

void main() {
    NestReplyStatus stat;
    NestConnection  server;
    NestLot         lot;
    NestLotHandle   lot_handle;

    stat = NestOpenConnection( &server, "nest.cs.wisc.edu" );
    stat = NestRequestUserLot( &lot_handle, server, 10, 1440 );
    stat = NestUpdateLot( server, &lot_handle, 0, 10 );
    stat = NestSendFile( "/usr/share/dict/linux.words", "words", server );
    stat = NestGetLotStat( &lot, server, &lot_handle );
}
```

Figure 1.5. **Nest-client library.** *This code sample, (which ignores error-handling for the sake of brevity), demonstrates the same functionality as shown using the interactive client in Figure 1.4. This code first connects to a nest server, then creates and updates a lot, writes a file to it and then queries its status.*

NeST, GARA provides reservations (similar to NeST's lots), but allows users to make them in advance. However, GARA does not provide the best-effort lots or the sophisticated user management that NeST provides.

The Disk Resource Managers in SRM (**?**), the storage depots in IBP (Plank et al., 2001) and the LegionFS servers (**?**) also provide Grid storage services. However, each of these projects is designed to provide both local storage management and global scheduling middleware. Conversely, NeST is a local storage management solution and is designed to integrate into any number of global scheduling systems. This distinction may account for one key difference between NeST and the storage servers in each of these systems: as they are all designed to work primarily with their own self-contained middleware, none of these other projects have protocol independence in their servers. Another unique feature of NeST is its dynamic concurrency adaptation; we note however that this is not intrinsic to the design of NeST and could be incorporated in these other systems.

SRM and IBP provide space guarantees in manners similar to NeST lots. One difference however in SRM is that SRM guarantees space allocations for multiple related files by using two-phased pinning; lots in NeST provide the same functionality with more client flexibility and control and less implementation complexity.

In comparing NeST lots with IBP space guarantees, one difference is that IBP reservations are allocations for byte arrays. This makes it extremely difficult for multiple files to be contained within one allocation; it can be done but only if the client is willing to build its own file system within the byte array. Another difference is that IBP allows both permanent and volatile allocations. NeST does not have permanent lots but users are allowed to indefinitely renew them and best-effort lots are analogous to volatile allocations. However, there does not appear to be a mechanism in IBP for switching an allocation from permanent to volatile while lots in NeST switch automatically to best-effort when their duration expires.

Like NeST, LegionFS also recognizes the importance of supporting the NFS protocol in order to allow unmodified applications the benefit of using Grid storage resources. However LegionFS builds this support on the client side while NeST does so at the server side. LegionFS's client-based NFS allows an easier server implementation but makes deployment more difficult as the Legion-modified NFS module must be deployed at all client locations.

Although NeST is the only Grid storage system that supports multiple protocols at the server, PFS (Thain and Livny,) and SRB (Baru et al., 1998) middleware both do so at the client side. We see these approaches as complementary because they enable the middleware and the server to negotiate and choose the most appropriate protocol for any particular transfer (*e.g.*, NFS locally and GridFTP remotely).

9. CONCLUSION

We have presented NeST, an open-source, user-level, software-only storage appliance. NeST is specifically intended for the Grid and is therefore designed around the concepts of flexibility, adaptivity, and grid-awareness. Flexibility is achieved through a virtual protocol layer which insulates the transfer architecture from the particulars of different file transfer protocols. Dynamic adaptation in the transfer manager allows additional flexibility by enabling NeST to run effectively on a wide range of hardware and software platforms. By supporting key grid functionality such as storage space guarantees, mechanisms for resource and data discovery, user authentication, and quality of service, NeST is grid-aware and thereby able to integrate cleanly with distributed computing systems.

10. Acknowledgments

This chapter is an updated version of the paper which appeared in HPDC 11. Some of the experimental results have been removed and replaced with a more functional description of using NeST to build grid services.
This work is sponsored by NSF CCR-0092840, CCR-0098274, NGS-0103670, CCR-0133456, ITR-0086044, and the Wisconsin Alumni Research Foundation.

References

Grid Service Specification. http://www.gridforum.org/ogsi-wg/.

(2003). *The Globus Project*. http://www.globus.org/.

Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L., Meder, S., and Tuecke, S. (2002). Gridftp protocol specification. *GGF GridFTP Working Group Document*.

Arpaci-Dusseau, A. C. and Arpaci-Dusseau, R. H. (2001). Information and Control in Gray-Box Systems. In *The 18th Symposium on Operating Systems Principles (SOSP)*.

Baru, C., Moore, R., Rajasekar, A., and Wan, M. (1998). The SDSC Storage Resource Broker. In *Proceedings of CASCON*, Toronto, Canada.

Bent, J. (2003). Nest storage appliance. Available from `http://www.cs.wisc.edu/condor/nest`.

Burnett, N. C., Bent, J., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. (2002). Exploiting Gray-Box Knowledge of Buffer-Cache Management. In *USENIX*.

Chiu, D.-M. and Jain, R. (1989). Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. In *Journal of Computer Networks and ISDN*, volume 17, pages 1–14.

Condor (2002). The Condor Directed-Acyclic-Graph Manager (DAGMan).

Crovella, M., Frangioso, R., and Harchol-Balter, M. (1999). Connection Scheduling in Web Servers. In *USENIX Symposium on Internet Technologies and Systems*.

EMC Corporation (2003). http://www.emc.com.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and Berners-Lee, T. (1997). RFC-2068: HTTP: Hypertext Transfer Protocol Specification Version 1.1. *Network Working Group Requests for Comments*.

Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S. (1998). A Security Architecture for Computational Grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security Conference*, pages 83–92.

Hitz, D., Lau, J., and Malcolm, M. (1994). File System Design for an NFS File Server Appliance. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 235–246, San Fransisco, CA, USA.

Howe, A. C. (2000). Bandwidth and Request Throttling for Apache 1.3.

Iyer, S. and Druschel, P. (2001). Anticipatory Scheduling: A Disk Scheduling Framework to Overcome Deceptive Idleness in Synchronous I/O. In *18th ACM Symposium on Operating Systems Principles*.

Kleiman, S. (1986). Vnodes: An Architecture for Multiple File System Types in Sun UNIX. In *USENIX Conference Proceedings*, pages 151–163.

Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. (2000). The Click Modular Router. *ACM Transactions on Computer Systems*, 18(3):263–297.

Lancaster, L. and Rowe, A. (2001). Measuring Real World Data Availability. In *Proceedings of the LISA 2001 15th Systems Administration Conference*, pages 93–100, San Diego, California.

Lohr, S. (2002). Supercomputing and Business Move Closer. New York Times Business/Financial Desk.

Pai, V., Druschel, P., and Zwaenepoel, W. (1999). Flash: An Efficient and Portable Web Server. In *Proceedings of the Usenix Technical Conference*.

Patterson, D. A. (2002). Availability and Maintainability $>>$ Performance: New Focus for a New Century. Key Note Lecture at the First USENIX Conference on File and Storage Technologies (FAST '02).

Plank, J., Bassi, A., Beck, M., Moore, T., Swany, M., and Wolski, R. (2001). Managing Data Storage in the Network. *IEEE Internet Computing*, 5(5).

Postel, J. (1980). RFC-765: FTP: File Transfer Protocol Specification.

Sandberg, R. (1985). The Design and Implementation of the Sun Network File System. In *Proceedings of the 1985 USENIX Summer Technical Conference*, pages 119–130, Berkeley, CA.

Sharpe, R. (1999). Just What Is SMB? samba.org/cifs/docs/what-is-smb.html.

Thain, D., Basney, J., Son, S.-C., and Livny, M. (2001a). The Kangaroo Approach to Data Movement on the Grid. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC 10)*, San Francisco, California.

Thain, D., Bent, J., Arpaci-Dusseau, A., Arpaci-Dusseau, R., and Livny, M. (2001b). Gathering at the Well: Creating Communities for Grid I/O. In *Proceedings of Supercomputing 2001*, Denver, Colorado.

Thain, D. and Livny, M. The Pluggable File System.

Waldspurger, C. A. and Weihl, W. E. (1995). Stride Scheduling: Deterministic Proportional-Share Resource Mangement. Technical Report MIT/LCS/TM-528, Massachusetts Institute of Technology, MIT Laboratory for Computer Science.

Walsh, D., Lyon, B., Sager, G., Chang, J. M., Goldberg, D., Kleiman, S., Lyon, T., Sandberg, R., and Weiss, P. (1985). Overview of the Sun Network File System. In *Proceedings of the 1985 USENIX Winter Conference*, pages 117–124.

Welsh, M., Culler, D., and Brewer, E. (2001). SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. In *In Proceedings of the Eighteenth Symposium on Operating Systems Principles (SOSP-18)*, Banff, Canada.

Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D. (1993). RSVP: A New Resource Reservation Protocol. *IEEE Networks Magazine*, 31(9):8–18.