IBM Storage Tank—A heterogeneous scalable SAN file system

by J. Menon
D. A. Pease
R. Rees
L. Duyanovich
B. Hillsberg

As the amount of data being stored in the open systems environment continues to grow, new paradigms for the attachment and management of data and the underlying storage of the data are emerging. One of the emerging technologies in this area is the storage area network (SAN). Using a SAN to connect large amounts of storage to large numbers of computers gives us the potential for new approaches to accessing, sharing, and managing our data and storage. However, existing operating systems and file systems are not built to exploit these new capabilities. IBM Storage Tank™ is a SAN-based distributed file system and storage management solution that enables many of the promises of SANs, including shared heterogeneous file access, centralized management, and enterprise-wide scalability. In addition, Storage Tank borrows policy-based storage and data management concepts from mainframe computers and makes them available in the open systems environment. This paper explores the goals of the Storage Tank project, the architecture used to achieve these goals, and the current and future plans for the technology.

IBM Storage Tank* (ST) is a multiplatform, scalable file system and storage management solution that works with storage area networks (SANs). By means of SANs, thousands of computers can connect to and share a large number of storage devices that range from simple disks to large, high-performance, high-function storage systems. The current state of SAN technology limits its use to machine room environments; therefore, ST is also currently limited in the same way. As SANs evolve beyond machine room environments, so will ST.

ST goes beyond cluster file systems, such as the IBM General Parallel File System (GPFS),[1] that allow a cluster of homogenous (single operating system) computers to share data by allowing thousands of heterogeneous computers, some subset of which may be clustered, to share data. ST can provide an effective solution for customers with as little as tens of computers and a terabyte of data, and can scale up to support customers with thousands of computers, petabytes of data, and billions of files.

In addition to sharing data, ST also centralizes storage management functions such as backup, restore, and file allocation. This centralization replaces the labor-intensive, computer-by-computer storage management that is currently in practice. ST further simplifies storage management by supporting policy-based storage management. An administrator specifies policies for how backup, restore, allocation, and so on are to be performed, and the ST system enforces these policies without human intervention. As a result, ST is an important step in the direction toward autonomic computing.[2]

0018-8670/03/$5.00 © 2003 IBM

This paper describes the Storage Tank architecture and design. Whenever we describe the current status of ST in this paper, we are referring to the status of the ST prototype built at the IBM Almaden Research Center.

Motivation

Customers face many issues today as they build or grow their storage infrastructures. Although the cost of purchasing storage hardware continues its rapid decline, the cost of managing storage is not keeping pace. In some cases, storage management costs are actually rising. Recent studies by Gartner, Inc.[3] and IDC[4] show that the purchase price of storage hardware comprises as little as 5 to 10 percent of the total cost of storage. Factors such as administration costs, downtime, environmental overhead, device management tasks, and backup and recovery procedures make up the majority of the total cost of ownership. Information technology managers are under significant pressure to reduce costs while deploying more storage to remain competitive. They must address the increasing complexity of storage systems, the explosive growth in data, and the shortage of skilled storage administrators. Furthermore, the storage infrastructure must be designed to help maximize the availability of critical applications.

To address these issues, ST provides centralized storage management for the many different servers and operating systems present in a typical customer environment. The alternative of individually managing many servers is a major reason for the high cost of storage management today.

In many customer environments, data sharing is also an important requirement. Data are often generated by an application running on a particular operating system and then later processed by other applications running on different operating systems. In some cases, this pipeline operation, from creation of data through final processing, may consist of three or four stages on three or four different operating systems. ST provides high-speed, direct-access data sharing across heterogeneous platforms for such applications, thus eliminating the need to copy data between the steps of the pipeline operations.

ST is designed to provide I/O performance comparable to that of local file systems, as well as increased scalability, high availability, and centralized, automated storage and data management. It is also designed to provide performance superior to network-attached storage (NAS) for data sharing among heterogeneous computers. These attributes will address customer requirements for growth capability, minimal-to-zero downtime, data sharing, and simplified storage management. The policy-based automation of storage and data management tasks in ST provides an important first step toward autonomic storage management.[2]

In the next section, the problems to be solved are described in greater detail. In the fourth section, we present the ST architecture and design, and in the fifth section, we compare the ST approach to alternative approaches. In the sixth section, we discuss our future plans for ST.

Problems to be solved

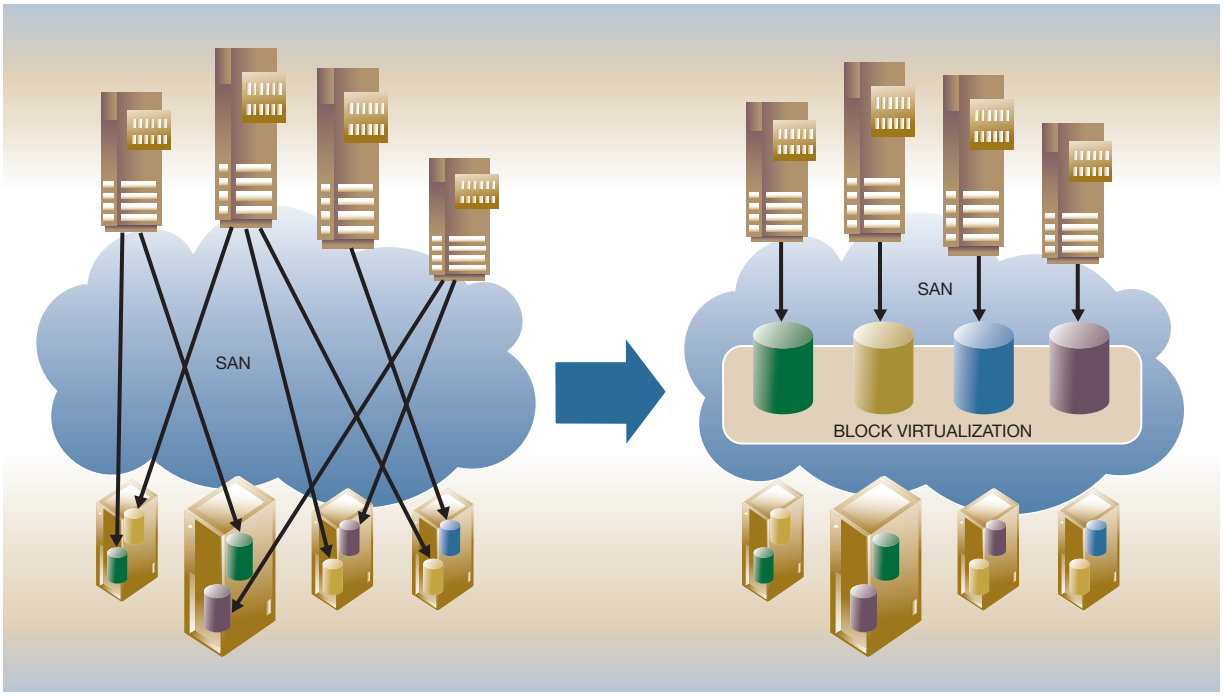
We begin with a description of storage area networks, or SANs. SANs solve many problems for customers and simplify storage management, but their full potential remains untapped without a scalable, multiplatform, SAN file system such as ST.

Background on storage area network technology. SANs enable the direct connection of large numbers of computers to large numbers of storage devices that range from simple disks to large, high-performance, high-function storage systems. A SAN can be built using Fibre Channel[5] networks, Ethernet networks,[6] or, in the future, InfiniBand** networks.[7] Computers use the SCSI (Small Computer System Interface) Protocol[8] to talk to storage systems or devices on the SAN—SCSI on Fibre Channel is called Fibre Channel Protocol, or FCP, SCSI on Ethernet is called iSCSI, and SCSI on InfiniBand is called SCSI VI Protocol (SVP). SCSI is a block-oriented protocol—computers access storage over the SAN by reading and writing blocks of data from storage devices.

With a SAN, storage can be separated from the computer that uses it. SANs make it easier for administrators to make server and storage decisions independently, and enterprises are not required to make their storage purchases from their system vendor. Large, high-performance, high-function storage systems, such as the IBM TotalStorage* Enterprise Storage Server*,[9] can be shared among many hosts.

For security purposes, SANs support zoning. Using zoning, we can group a subset of the computers and a subset of the storage systems into a zone so that they can see only each other, not other computers and storage systems in other zones.

Figure 1 Virtualization



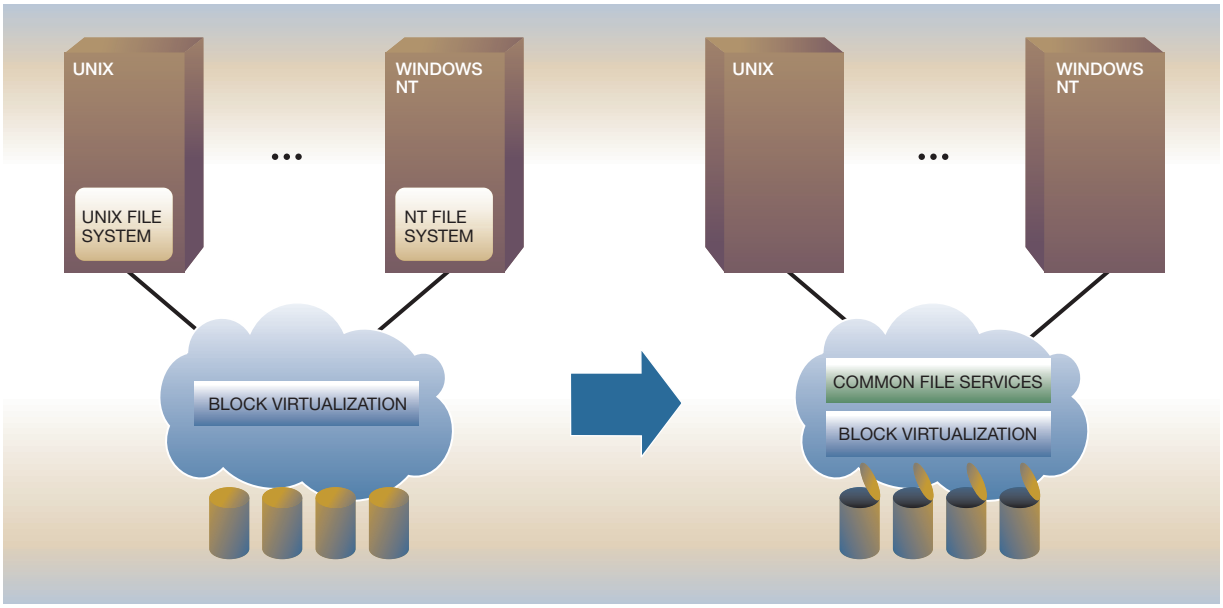
Furthermore, storage space can be assigned where it is needed. In a traditional environment, unused logical volumes or partitions on one system typically cannot be used by other systems. With a SAN, that free space can be partitioned and allocated efficiently and effectively. The ability to share free space among multiple systems allows administrators to achieve higher space utilization levels and thus lower their overall storage costs. However, although SANs and the storage devices that support them have provided improvements in storage and space management, many of the opportunities presented by SANs remain untapped.

Background on virtualization. Essentially, virtualization introduces another layer between computers and storage systems. This layer permits a level of indirection between storage devices as seen by computers and storage devices as exported by storage systems. Virtualization is illustrated in Figure 1. Virtualization allows a storage device as seen by a computer to actually span multiple storage systems. The virtualization layer keeps the mapping between storage devices as seen by computers (host storage devices) and storage devices as exported by storage systems (exported storage devices). Virtualization allows remapping of host storage devices to exported storage devices without causing a computer to be rebooted—as a result, it deals more gracefully with the addition and removal of storage systems than systems without virtualization.

Problems still unsolved. SANs and virtualization allow storage devices to be shared by multiple heterogeneous operating systems. However, native file systems, such as NTFS (New Technology File System) for the Windows** platform[10] or JFS (journaled file system) for the AIX* operating system,[11] expect to have exclusive access to their volumes, and data in these file systems cannot be directly shared with other systems.

There are several other problems. First, each different operating system, for example the UNIX** and Windows systems, reserves storage devices for its own use, and space in a storage device owned by one operating system cannot be used by another. Second, a computer cannot create a file anywhere on any storage device on the SAN; it is limited to storage devices or virtual disks that it owns. Third, a computer

Figure 2 Difference between a SAN with virtualization and a SAN that uses virtualization with ST



does not have access to all files created by any computer, but only to the files it creates. Fourth, it is not possible to move an application from a computer running one file system to another computer running a different file system without also reading all of the data of the application, modifying the data to the format of the second file system, and writing all the data back to disk. Note that some of the problems mentioned above may exist even if the different computers are running the same operating system.

All of these problems are addressed by ST. As a result, ST offers the potential to simplify storage management significantly.

Figure 2 shows the difference between a SAN with virtualization that uses many local file systems, one on each computer (on the left), and a SAN that uses virtualization with ST to provide a common set of file services for all computers (on the right).

A variety of shared file systems that provide common file services, such as Network FileSystem (NFS**),[12] Common Internet File System (CIFS),[13] and the distributed file systems AFS*/DFS*[14] have existed for a number of years. Typically, such shared file systems have been subject to three limitations. First, some of these file systems do not appear like a local file system to applications that share data, because they provide semantics that are subtly different from those expected of a local file system. Second, they interpose an additional computer between the computer that wants to access the data and the storage system or device. This interposition causes performance bottlenecks, particularly for data-intensive applications. Third, they often do not provide a global namespace automatically; that is, without significant administrative burden, they do not provide the ability to access a file from any computer using the same name. With ST, we avoid all of these problems.

Another alternative shared file system is a cluster file system, such as GPFS.[1] Cluster file systems can provide local file system semantics and do not interpose an additional computer in the path to data. However, current cluster file systems support only homogeneous environments and, therefore, do not satisfy our heterogeneity requirements. Furthermore, they are focused on high-speed data sharing and do not address the requirements for simplified storage management.

The fifth section of this paper compares ST to other shared file systems in detail.

Storage Tank goals and architecture

We are now ready to describe ST more fully. We begin by enumerating the goals for the ST project and then describe the ST architecture. In reading Storage Tank project goals, keep in mind that ST has a client/server architecture (as shown later in Figure 3). Note that the servers are configured in a cluster for scalability and availability and manage only meta-data. ST clients access data directly from storage devices attached to a storage area network.

Storage Tank project goals. The goals of the ST project are as follows:

- Exploitation and enablement of SAN technology—Traditional network file systems, including NAS, use a client/server data access model that interposes intermediate computers called servers in the path to data. They have the additional limitation of using conventional networks to transfer the data. Although these systems allow users to share data, they do not provide the performance required for data-intensive applications. In contrast, ST uses a data access model that allows computers to access data directly from storage devices using a high-bandwidth SAN, without interposing servers. Direct data access helps eliminate server bottlenecks and provides the performance necessary for data-intensive applications.

- Heterogeneous (multiplatform) data sharing—The ST file system is specifically designed to be easy to implement in virtually any operating system environment. All systems running this file system, regardless of operating system or hardware platform, have uniform access to the data stored in the system. File meta-data, such as directory entries, and file attributes, such as last modification time, are presented to users and applications in a form that is compatible with the native file system interface of the platform. ST servers provide the locking needed to ensure consistent data sharing between heterogeneous computers. Our goal is to make ST a solution that is superior in performance to NAS for heterogeneous data sharing.

- Policy-based storage and data management—One of the most important goals of the ST project is to reduce the effort and cost associated with managing large, complex storage environments. A key to achieving this goal is policy-based storage and data management. ST provides the storage administrator with tools for automating the management of storage resources and the data stored on those resources. An example of policy-based management in ST is the automatic placement of files on appropriate storage devices based on specified application performance requirements. Policy-based management of data is a concept that has been common in IBM mainframe (z/OS*) installations for decades;[15] ST brings these desirable features to all operating systems.

- Massive scalability of data, servers, and clients—The amount of data being stored and managed by installations is growing at an unprecedented rate. A system such as ST must be ready to manage ever-increasing amounts of data and numbers of computers. ST achieves scalability through its ability to:

  – Attach and control massive amounts of storage (petabytes)
  – Efficiently manage huge numbers of files (billions)
  – Connect to and control file access for large numbers of clients (thousands)
  – Add as many servers to a server cluster as needed to support the client load and efficiently distribute work among those servers

  Our goal is to make ST useful over a wide range of customer environments. Although ST is designed to support massive scalability, it is also designed to be an attractive solution for smaller customers with much smaller numbers of computers and less data. Thus, small customers can install ST, and ST will be able to grow with them as they accumulate more data.

- High availability of servers and data—An ST server cluster performs load balancing and fail-over processing to ensure that data are available to users quickly and continuously in the event of a server failure. When a server fails, ST reallocates its workload to other servers in the cluster.

  ST is also designed so that an administrator can perform SAN administration tasks, such as adding or deleting disks, allocating storage, moving data, or initiating a backup, without interrupting system availability. This design allows execution of storage administration tasks while all ST clients and servers remain operational and all data remain on line and available to users and applications.

  Finally, new ST clients can be commissioned without reconfiguring or otherwise interrupting the running system.

- Global namespace and single-system semantics—The ST system presents a single namespace view of all files in the system to all of the clients, without manual, client-by-client configuration by the administrator. As a result, a file can be identified using the same path and file name, regardless of the system from which it is being accessed. Furthermore, ST organizes its global namespace such that clients need not specify the actual location of the data (e.g., the name of the server that serves the data). This is called *location independence*.

  ST differs from previous distributed storage systems by providing exact local file system semantics to its clients. ST lock semantics are rich enough to fully describe and enforce local file system semantics in a distributed environment. When clients or servers fail, ST is designed to enforce cache coherency and proper file system semantics across the remaining systems.

- Multiple server platforms—The ST server is implemented as a highly portable, user-level application, allowing it to run on many systems from low-end Linux** personal computers to high-performance SP2* supercomputers, or even mainframes. Server portability gives an administrator the flexibility to choose the appropriate server platform for the needs of an enterprise. The ST server has been run on Linux, several versions of UNIX, and Windows NT**, Windows 2000, and Windows XP.

- High-performance file I/O for all kinds of files—Because ST accesses data directly over the SAN, all file I/O operations can be as fast as those using a local file system. Features such as striping and parallel I/O operations can allow ST to serve applications with I/O bandwidth requirements beyond what is typically provided by the native file system. ST is designed to perform well for all kinds of files, both small and large. ST improves performance for large files by eliminating the server from the data path, and it achieves good performance for small files by using more sophisticated caching of meta-data and through the use of a protocol that reduces message traffic relative to protocols such as NFS.

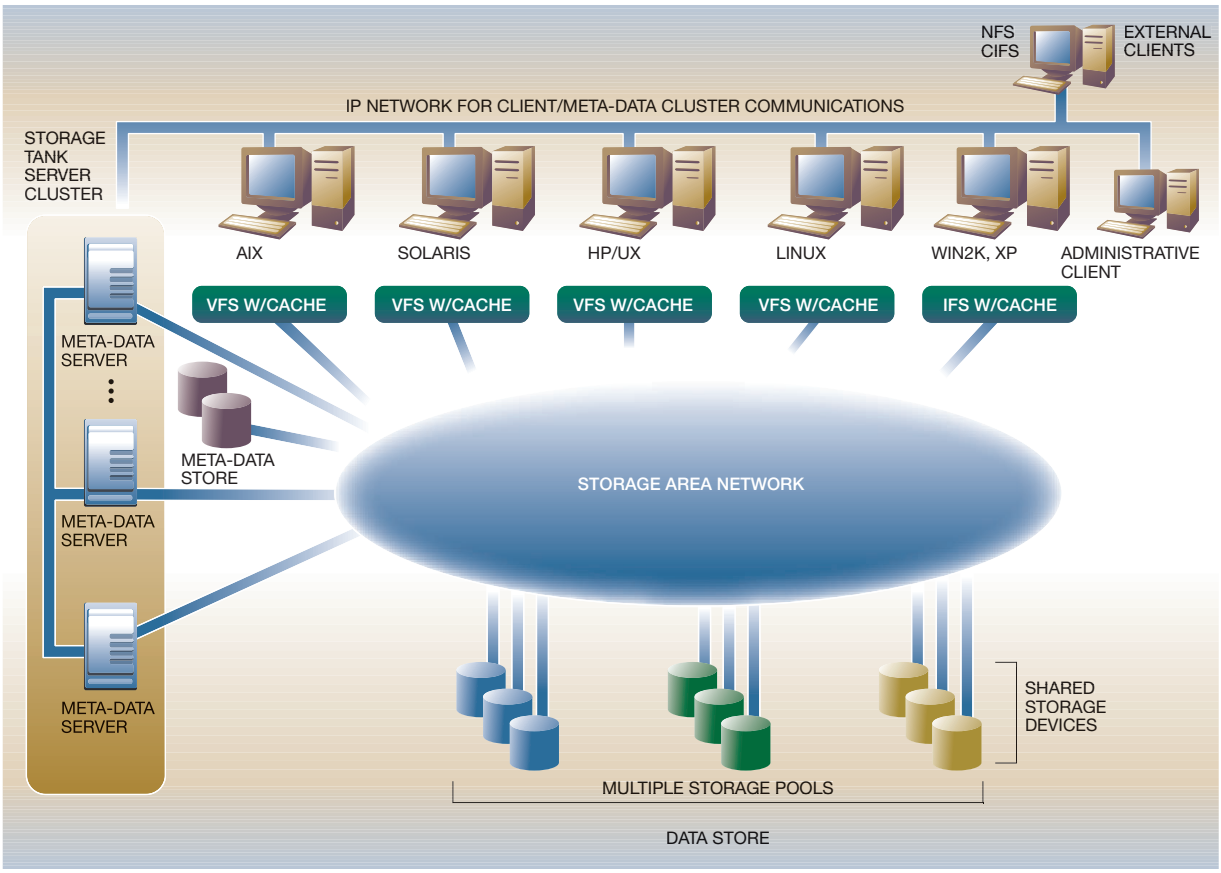
Storage Tank architecture. Figure 3 illustrates the Storage Tank architecture. Computers that want to share data and have their storage centrally managed are all connected to the SAN. In Figure 3, we show five such computers, each running a different operating system. ST is designed to support thousands of computers, initially running these five different operating systems, and then other operating systems over time. Each of these computers will run a program called the *ST client*. Since the ST client is inserted at the virtual file system (VFS) interface on UNIX systems and as an installable file system (IFS) on Windows systems, the clients are shown as VFS or IFS in the figure.

With ST, computers that want to access data run the ST client. There are also special computers called *ST servers*[16] that run ST server code, as shown on the left side of the figure. The ST servers manage file system meta-data (file creation time, file security information, file location information, and so on), but data transferred from storage systems or devices to computers do not need to pass through the ST servers. This eliminates the performance bottleneck problem from which many existing shared file system approaches suffer. ST servers are clustered for scalability and availability of meta-data operations and are often referred to as an *ST server cluster*. Storage systems and devices that store computer data are separated from storage systems or devices that store ST meta-data, as shown in the figure.

ST is built on two logical networks. The control network is used by file system clients and the administrative client to communicate with ST servers. This control network carries only lock state and meta-data, so the amount of data transferred over it is minimal. The control network is implemented on an Internet Protocol (IP) network using the IBM Storage Tank Protocol.[17] The second network is the SAN. ST clients, servers, and storage devices are all connected to the high-speed SAN. The SAN is used for all data transfer, which removes the ST server from the data path and eliminates performance overhead and potential bottlenecks. ST is designed to be independent of the actual SAN fabric technology. It works with Fibre Channel networks,[5] as well as new emerging storage networking technologies such as Gigabit Ethernet (iSCSI)[6] and InfiniBand.[7]

The ST administrative client serves as the administrative control point. An administrator can perform administrative tasks on line with no service interruption to applications running on the clients. An IFS (or VFS in the case of supported UNIX clients) is installed on each ST client. An IFS directs requests for meta-data and locks[18] to an ST server and sends requests for data to storage devices on the SAN. ST clients can access data directly from any storage device attached to the SAN. ST clients aggressively cache

Figure 3 IBM Storage Tank system architecture



file data, as well as meta-data and locks that they obtain from an ST server, in memory. In contrast to some other distributed file systems, there is no need to cache files to local disk because ST clients already have direct access to data on disk.

Unlike most file systems, ST stores meta-data and data separately. Meta-data, which include standard file meta-data such as file name, creation date, and access control information, also contain the location of the file data on disk (the extent list). Separating the meta-data and data allows meta-data to be stored on high-performance, highly available private server storage (which can be on the same SAN as the data storage or on a separate SAN) if needed. Meta-data must be accessible by all servers in the cluster. Meta-data are never directly accessed by clients but are served via the ST Protocol over the IP control network.

Data blocks for any given file are stored on data disks. Data disks must be configured on the SAN to be accessible by both ST clients and ST servers. In most situations, the SAN would be configured with one zone for ST data disks, clients, and servers. It is also possible, if desired, to create zones accessible by only the ST servers and a subset of ST clients to meet special security requirements.

An enterprise can use one ST server, a cluster of ST servers, or multiple clusters of ST servers. Clustered servers provide load balancing, fail-over processing, and increased scalability. The servers in a cluster are interconnected, either on their own high-speed network or on the same IP network that they use to communicate with ST clients. The private server storage that contains the meta-data managed by a server cluster can be attached to a private storage network, perhaps in a separate zone.

Storage Tank abstractions. In addition to the traditional abstractions found in file systems such as files, directories, and disk volumes, ST also defines the following two primary abstractions.

- Container—A container is a subtree of the global namespace. It groups a set of ST files and directories for the purpose of load balancing and management. After an administrator defines appropriate containers, ST automatically performs load balancing of these containers across the ST meta-data servers. Containers are also the basic unit for storage management operations such as backup, recovery, and point-in-time snapshot. A system can have many containers, each capable of storing on the order of 100 million files, allowing ST to scale to billions of files.
- Storage pool—A storage pool is a collection of one or more disk volumes. It provides a logical grouping of the volumes for the allocation of space to containers. It is expected that different storage pools will have different performance and availability characteristics. An administrator can specify policies that allow different applications to automatically choose different storage pools for the placement of their data. The files in a container can belong to different storage pools. Multiple containers can own storage within a single storage pool.

Together these two abstractions allow independence between logical naming and physical placement. This is unlike most file systems, in which the path name of a file determines the logical volume on which it is stored.

Storage Tank Protocol. The Storage Tank Protocol is used for communications between ST clients and ST servers.[17] The protocol implements a locking and data consistency model that allows the ST distributed storage system to look and behave like a local file system.[18] The objective of the ST Protocol is to provide strong data consistency between clients and servers in a distributed environment in which a SAN is used to transfer data between clients and disks.

The ST Protocol allows aggressive caching at the client and allows the retention of modified meta-data, data, and lock state beyond close-of-file by an application. Furthermore, allocation operations are piggybacked on other protocol messages for efficiency.

The ST Protocol provides locks that enable file sharing among ST clients or, when necessary, provides locks that allow clients to have exclusive access to files. An ST server grants locks to clients when files are opened. The ST Protocol guarantees that when a client reads data from a file, it always reads the latest data written to that file by any client.

The ST Protocol includes support for recovery from network partitions and from server failures within the server cluster through the use of leases.[19] It also provides copy-on-write capability for snapshots or point-in-time copies of data in a SAN environment.

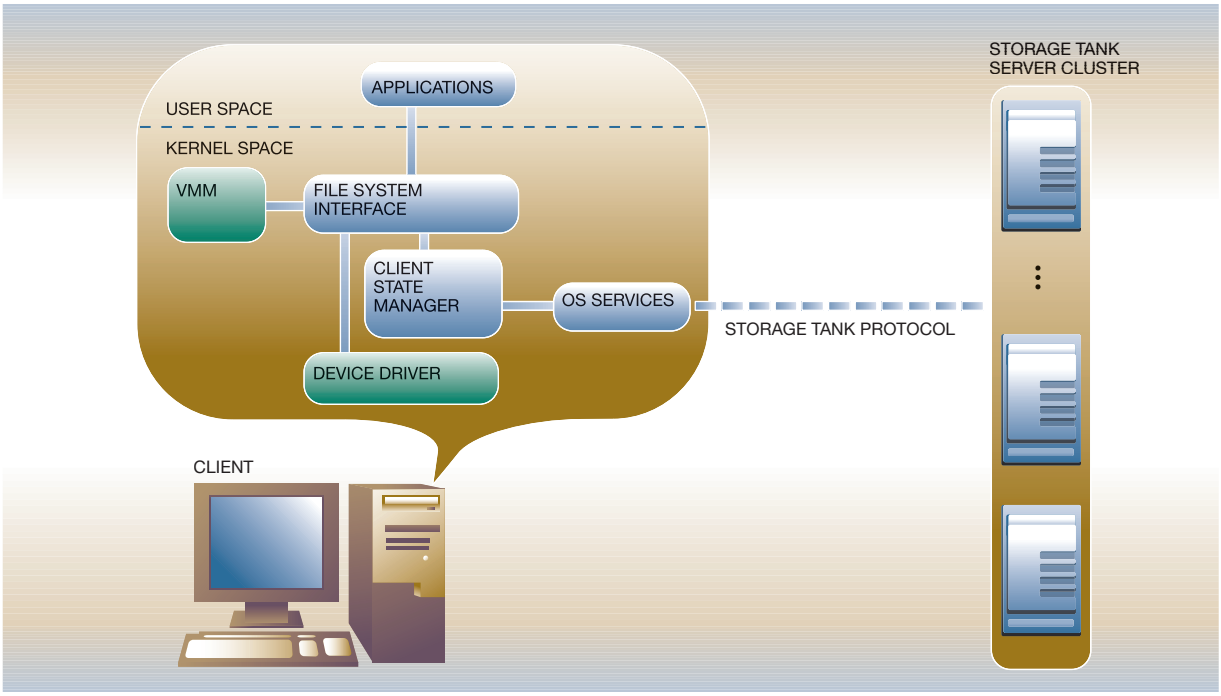
Storage Tank clients. ST enables full, transparent data sharing of files among heterogeneous clients, such as those running the Windows 2000, AIX, Solaris**, Linux, and HP-UX** operating systems. All ST clients can access the same data using the ST global namespace as described before. This capability requires no application changes; applications use the same interfaces to access ST data as they use to access the native file system.

The ST client is designed to direct all meta-data operations to an ST server and direct all data operations to storage devices attached to a high-speed network. It makes the meta-data that are visible to the client's operating system (and to any applications running on the system) look identical to meta-data read from a native, locally attached file system.

An ST client is composed of three components as shown in Figure 4: the file system interface, the client state manager (CSM), and operating system services. Porting the ST client to a new operating system involves writing the platform-specific file system interface and operating system services. The CSM, which incorporates the ST-specific part of the file system, is platform-independent and does not need to be changed. The installable file system makes use of the native virtual memory management and device drivers of the platform. The client end of the ST Protocol is implemented by the CSM. The CSM is the intermediary between the platform-specific client file system and the ST server.

The CSM maintains both session locks, which are acquired at file open and are needed to ensure session semantics, and data locks, which are needed to ensure the consistency of the data and meta-data. We chose to separate locking for sessions on files (open files) from locking for cache consistency. This separation permits more flexible caching policies; for example, a file can be prefetched into cache prior to any application opening that file.

Figure 4 Storage Tank client architecture



The ST client makes an ST file system appear to be just another file system on a client. A user sees no difference between accessing a file from the ST file system and accessing a file from a local file system. For example, to open a file in the ST file system, an application issues a standard file open request. The client file system passes the request to the CSM, which determines whether the request can be satisfied using locks already in its cache. If not, the CSM contacts the ST server to obtain the file meta-data and locks. The file meta-data supply the client with information about the file—its attributes and location on storage device(s). Locks supply the client with the privileges it needs to open the file and read or write data.

Read and write requests must also be passed to the CSM to ensure that locks are consistent with the access requests. If, for example, the request is a write, but the lock is valid only for reading, the CSM communicates with the server to request that a lock be upgraded. Once the required lock or locks have been obtained, file data can be accessed directly over the SAN.

ST session locks can be cached or semi-preempted,[18] allowing file system clients to retain distributed locks even when there are no open instances of the file. Thus, requests for a given file by subsequent applications at the same client may be able to be satisfied without incurring the overhead of contacting the server and obtaining new locks. If a client requests an open mode for a file that is incompatible with a lock already cached at another client, the ST server asks the other client whether it can comply with the requested open mode. If there are no incompatible open file instances, the client complies. Otherwise, the requesting client is denied access.

The ST compatibility matrix for open modes is complex and is designed to capture the many open modes defined by both the Windows and UNIX operating systems. Details are beyond the scope of this paper. However, our scheme supports both the read and write compatibility on UNIX and the exclusive modes of Windows. As an example, whether open for write is compatible with open for read is a function of the specific open mode requested.

The CSM also implements the client side of a lease-based protocol that protects the system from consistency errors caused by network failures.[19] The ST server maintains a lease for each client in the sys-

tem. This lease is opportunistically updated with each client/server interaction. If the server is unable to renew the lease with that client, that client is assumed to have failed, and the server sets a timer. At the end of the set period, the server recovers the locks and is free to provide them to new clients. Leases allow the system to make forward progress in the event of network failures when failed clients hold locks needed by other clients. We expect to exploit hardware fencing capabilities in SANs to isolate lease-expired clients.

From the client side, when the lease expires, the client must write all "dirty" data from its cache to disk. Access to the client's cached data is suspended until the lease is resolved.

The ST client cache is used to achieve low-latency access to meta-data and data. A client can cache the following:

- Data—Caching data allows a client to perform reads and writes for smaller files locally, potentially eliminating I/O operations to SAN-attached storage devices.
- Meta-data—Caching meta-data allows a client to perform multiple meta-data accesses locally without contacting an ST server.
- Locks—Caching locks allows a client to grant multiple opens to a file locally without contacting an ST server.

An ST client performs all caching in memory. If there is not enough space in the client's cache for all of the data in a file, the client simply reads the data from the shared storage device on which the file is stored. Data access is fast because the client has direct access to all storage devices attached to the storage network. There is no need for a client to cache data to a private disk.

Storage Tank servers. The ST server is a portable, user-level, C++ application that is easily moved to new operating systems. "Ports" have been done for the Linux, AIX, Solaris, and Windows operating systems.

Support for multiple operating systems provides flexibility in choosing the meta-data server cluster platform and allows a range of performance options. For example, Intel processors running Linux could be used for cost-effective scalability, whereas an IBM SP2 supercomputer running AIX could be used for high-end scalability.

ST servers provide meta-data, storage management, and data management services. The following subsections provide detailed information about each type of service.

Meta-data services. An ST server is designed to perform meta-data updates, serve file system meta-data to clients, grant file and data locks to clients, and detect client failures and perform client recovery. It is fundamentally a shared-nothing cluster that is optimized for a file system workload. Unlike NAS servers, ST servers perform no data operation. Therefore, the ST server is designed and tuned to handle a meta-data workload—a workload with many operations on small objects that are short in duration, and in which there are ordering constraints. It implements the ARIES write-ahead logging algorithm[20] for transaction management, memory management, and recovery. In normal operation, servers never need to communicate with each other, allowing the design to be very scalable. Servers in an ST server cluster need to communicate only with each other during failure detection and recovery.

To enhance performance, ST servers write log records asynchronously to disk. To preserve the structural integrity of the file systems, ST ensures the following ordering: data are first written to disks on the SAN; the meta-data are then sent to the server for updating; finally, the free-space data structures maintained by ST are updated.

An enterprise can use a single server, a cluster of servers, or multiple clusters of servers. Multiple clusters of servers may be used when the need for separate administration domains within an enterprise arises. Using ST servers in a cluster configuration has the following benefits:

- Load balancing—The containers defined by an administrator are the units used for automatic load balancing among the servers in a cluster. Assigning different containers to different servers permits the server cluster to operate with minimal synchronization.
- Fail-over processing—ST servers implement a clustering protocol. In the event of a server failure or loss of network connectivity between servers, the cluster services cause a new cluster to be reformed, and the load is distributed among the servers in the new cluster. The ST group service detects the loss of a server and assigns another server to recover the log of the failed server, clean up the log, and redistribute the workload of the failed server

to other servers. Clients of the failed server are redirected to the other servers. The ST Protocol allows clients to reassert the locks[19] that they held with the failed server with the other servers in the cluster. In this way, the servers do not have to know which locks were being held by the failed client. By reasserting locks, clients can retain previously cached data, and they avoid the expense of obtaining new locks and reading data from storage devices again.
- Scalability—An administrator can add more servers to a cluster or add more server clusters to the SAN to serve more data and more clients. The clustering services described above detect a new server, form a new group that includes the new server, and redistribute load to balance work across all servers in the new group automatically and with no application impact.

To avoid the processing overhead of dealing with differences between server platforms, the ST servers in a specific cluster must currently all be of the same type. However, an installation can deploy multiple clusters of different types. For example, an enterprise might have one server cluster in which all the servers run AIX, and another server cluster in which all the servers run Linux.

Administrative services. There can be one or more administrative clients connected to a server cluster. These clients run an ST-specific administration application program and are used by administrators to manage the ST environment. An administrative client is connected to the ST servers via an IP network. To perform administrative tasks, an administrator can choose to use either a graphical user interface or a command line interface.

From an administrative client, an administrator can perform system management, storage management, and data management activities. The ST design allows the following administrative tasks to be performed on line with no interruption in service to clients:

- Create an ST server cluster.
- Commission or decommission server nodes.
- Create and maintain storage pools based on quality of service or other enterprise-specific requirements. For example, an administrator could create a storage pool that consists of RAID (redundant array of independent disks) or striped storage devices to meet reliability requirements, and could create another storage pool that consists of cached storage devices to meet high-performance requirements.
- Manage storage devices; for example, an administrator can add or remove disk volumes.
- Create containers that are subtrees of the directory tree in the namespace of a cluster.
- Manage quotas that control the size of containers or the portion of any given storage pool that can be used by a particular container.
- Create or manage file placement and other policies (see the subsection on data management services).
- Invoke data movement services (see the following subsection).
- Create snapshots by container (see the subsection on data management services).

Storage management services. ST is designed to perform a variety of storage management services based on storage management policies set up by an administrator. It performs these services across the SAN with no client involvement, either automatically or at an administrator's request. The following are among the storage management services provided by ST:

- Storage pool monitoring, including alert generation when utilization thresholds are exceeded.
- Volume decommission, which allows data to be moved from a disk volume to other free space in the storage pool when that volume must be removed from service. This function can be used when equipment leases are ending, or to move data from a volume that is experiencing a high error rate. Applications running on ST clients are not disrupted during the data movement.

Data management services. An ST server is designed to perform these data management services:

- Select a service class for each file based on policies set by the administrator. Any file attribute known at file creation time, such as file name, user, group, or client name, can be used to select the service class for a file. The service class describes the requirements for the file and includes attributes such as allocation options, caching options, performance priority, and availability requirements.
- Manage allocation and placement of data in storage pools on storage devices. Policies are set by the administrator. Any file attribute known at file creation time or the service class can be used to select the storage pool in which the file is stored. If there is no room in the selected storage pool,

files "spill" into the default storage pool, and that event is logged.
- Use policy-based backup and policy-based file retention, similar to the policy-based file placement described above.
- Move data between storage pools. This function provides a nondisruptive data movement capability for a file or set of files. It is useful for moving spilled files into their correct storage pool once space has been made available or for correcting storage pool assignments caused by incorrect policies.
- Take snapshots of the ST data and associated metadata. A snapshot creates a point-in-time view of a container. ST clients support snapshots by implementing a copy-on-write facility. Any data a client writes to a file following a snapshot are written to a new location in storage. The data that existed when the last snapshot was taken remain in their original location. Data mining applications and backup utilities can access the point-in-time view of ST data without interrupting normal data operations in the file system. While a snapshot is being taken, all data remain on line and can be read and written by clients.

Use of storage pools. An administrator can choose to use various types of storage on the SAN. Data storage can be any SAN-enabled disk or subsystem (such as RAID, JBOD [just a bunch of disks], or hierarchically managed devices) and will ultimately include tape and optical devices.

A storage pool can consist of multiple disks that reside on any combination of heterogeneous storage devices. To an application, a storage pool appears as a single storage space in which the application can store data without the need to know anything about the characteristics or boundaries of the physical disks.

A storage administrator sets up storage pools to meet specific needs of an enterprise. For example, an enterprise might want to deploy storage pools that consist of disks located on fast devices for transactional data and storage pools of disks on slower devices for backup data. An enterprise might also want to deploy multiple storage pools that provide different availability characteristics, or, perhaps, separate storage pools for each department within the enterprise.

After setting up storage pools, an administrator can increase or decrease the sizes of specific storage pools to meet changing needs and can easily move data from one storage pool to another. All of these tasks are transparent and nondisruptive to users and applications.

SAN/NAS convergence using Storage Tank. In general, we believe that ST can be superior to NAS in many respects. Nevertheless, there can be situations where a customer also needs NAS access. For example, a customer has a SAN in the machine room but also needs to access data from other machines that are outside of the machine room. The customer chooses NAS access for the machines outside of the machine room. In a second example, the customer chooses to implement a small SAN to connect only his or her high-performance servers to storage. For all other servers, the customer decides that the lower performance of NAS is acceptable. In a third example, the customer may have some machines running operating systems that are not supported by ST and, therefore, chooses NAS to connect these machines to shared storage.

A storage pool can consist of multiple disks that reside on any combination of heterogeneous storage devices.

The problem is that most vendors provide different SAN and NAS storage solutions. This is inflexible for customers because it does not let them share data between servers connected to the SAN and servers connected to NAS. Furthermore, as customers convert servers using NAS access to use SAN access, data will need to be moved from NAS storage to SAN storage. Besides the disruption to the applications during this data move, the move might also idle the NAS storage resources, while possibly forcing customers to make new SAN storage acquisitions.

In our view, choosing between SAN access and NAS access should not be forced on customers at the time they acquire the storage solution. In our approach, customers acquire SAN storage and Storage Tank for all of their needs. Servers that need NAS access can obtain access via a gateway machine running a NAS server (such as NFS or CIFS) along with an ST client. Data can be created using SAN access from one server and read back later using NAS access from another server, or vice versa. It is also possible to have multiple gateway machines that allow access to a com-

mon set of files stored on ST by running a distributed version of NAS server code in the multiple gateway machines. The distributed NAS server needs to coordinate NAS locking and NAS client cache control across all the gateway machines for correct operation. The following are some other benefits of using such a gateway approach:

- Scalability—A single NAS gateway can access all of the files in ST. Since ST is designed to be more scalable than many other file systems, such an approach allows access to large amounts of data. Furthermore, a distributed set of NAS gateways can be built to support a very large number of NAS clients.
- One hundred percent compatibility with both NFS and CIFS—Typically, NAS solutions offer full NFS compatibility and less than full CIFS compatibility, or they offer full CIFS compatibility, but less than full NFS compatibility. By running an NFS server and ST client on a UNIX gateway, and a CIFS server and ST client on a Windows gateway, ST allows full compatibility of both NFS and CIFS. This is possible because the underlying ST file system supports both the Windows and UNIX operating systems.
- Reliability and fail-over processing—Because many NAS gateways can be ST clients and can export the same files, requests from clients of a failed NAS gateway can be transferred to another NAS gateway using any technique supported by the file server, such as high-availability cluster multiprocessing or IP address stealing.
- Policy-based storage management for NAS access.

To summarize, customers can be given two choices for accessing data in ST. They can load the ST client on a computer and then access the data using the SAN directly, thus allowing high-speed access to ST data. Alternatively, customers may choose not to load the ST client on a computer. From this computer without the ST client, access to ST data can still be made using a NAS protocol such as NFS or CIFS. That request will arrive at a NAS gateway server running an ST client. Sharing of data between computers with an ST client installed and computers without ST clients installed is now possible.

Support for other applications. Storage Tank provides support for database systems and other applications.

Database support. The ST file system provides a special mode of operation, called the *passthru I/O mode*, which is particularly useful for database systems and for applications that perform sequential I/O operations. When using this mode, the system performs direct writes to disk, does not cache data, and allows distributed applications on different computers to write data to the same file at the same time. Using the passthru I/O mode makes files behave more like raw devices. This gives database systems direct control over their I/O operations, while still providing all the advantages of the ST file system, such as policy-based management and volume backup.

Since database systems do their own caching, this mode of operation eliminates double caching. It also eliminates the false sharing that results when databases want to operate against different tuples that are stored in the same disk block.

File system caching also hinders applications that perform sequential I/O operations and does not provide them much benefit. Thus, we expect that passthru I/O mode may also be beneficial for high-performance computing applications.

The passthru I/O mode is associated by policy with a database file at file creation.

ST will be enabled to support the DB2* (Database 2*) Data Links Manager,[21] which provides referential integrity between databases and file systems. This will prevent files referenced by databases from being moved or deleted, thereby maintaining referential integrity.

Other application support. Special-purpose applications, for example, digital libraries, can access data from the ST distributed storage system by using an application programming interface to talk directly to the ST server using the ST Protocol. Because these applications do not use the file system to access their data, the clients on which they run do not need to have the IFS installed.

Comparison of Storage Tank to alternative approaches

ST provides a shared file system for servers in a machine room. In this section, we compare it to alternative technologies that address similar needs.

Comparison to NAS. One popular technology for providing shared access to file data is NAS. NAS products are available from IBM, Network Appliance, Inc.,[22] EMC Corporation, and several other vendors. All file data to be shared are stored on a NAS server,

and machines that want to access data do so through the NAS server.

The NAS approach is inferior to ST in several ways that we have articulated elsewhere, but repeat here. First, such an approach is slower than ST because all data must pass through an extra server (the NAS server). Thus, it impacts throughput for data-intensive workloads. It may also impact latency. In addition, the ST Protocol permits a more sophisticated level of caching and requires fewer messages between the client and the server, which can provide response time benefits. Our experiments show that NAS can be as much as three to ten times slower than using a local file system today. In the future, with the availability of network cards that offload TCP/IP (Transmission Control Protocol/Internet Protocol) processing, and the use of remote DMA (direct memory access) capability, we expect that NAS will become much faster. However, our analysis leads us to believe that NAS will continue to need 1.2 to 2.5 times as much processing power as an approach using local file systems and SAN storage to achieve equivalent performance.

There are many important differences between ST and NAS accelerators.

Second, NAS semantics, particularly NFS semantics, are different from local file system semantics for applications that do data sharing. New versions of NFS, like NFSv4, come closer to local file system semantics, but differences remain. An application written to use an NFS server cannot expect the same locking and consistency guarantees (for example, when there are multiple applications writing to the same file) obtained from a local file system. In contrast, the closeness of ST semantics to local file system semantics is important for applications that perform data sharing.

A third difference is that NAS systems typically do not provide a global namespace automatically, whereas ST does. As a result, when an ST server is added, there is no impact on applications. When a NAS server is added, however, there is typically a manual movement of data needed between the NAS servers, and all clients must remount the NAS servers.

AFS/DFS,[15] like NAS, interposes an extra server between the computer and the storage system or device. However, it is better than NAS in that it does provide a global namespace, and it does come much closer to achieving local file system semantics.

Comparison to DAFS. The NAS vendors, Network Appliance in particular, appreciate the shortcomings of NAS for use in machine room environments. They have proposed a new protocol called Direct Access File System (DAFS)[23] as an alternative to NAS.

DAFS semantics are closer to local file system semantics, eliminating one of the issues with NAS. DAFS also has improved performance relative to NAS. However, it still requires the use of an additional server in the path to the data, so we believe that its throughput and response time will still not be as good as that of ST, even though DAFS clients may consume fewer CPU resources than ST clients.

Furthermore, DAFS requires the use of a new kind of storage network—one that has Remote DMA capability. In particular, DAFS requires the deployment of a new kind of network interface card. Most storage networks deployed by customers today do not support Remote DMA; therefore, they cannot support DAFS unchanged. ST, in contrast, can work with existing SAN networks that customers already have in place.

Several varieties of DAFS have been proposed. The version of DAFS that provides the most significant performance improvement, called uDAFS, requires changes to applications in order to realize its benefits. But ST requires no application changes.

Comparison to NAS accelerators. Another class of product to which we need to compare ST is NAS accelerators. Examples of NAS accelerators are SANergy* from IBM and HighRoad** from EMC. NAS accelerators work just like NAS, except when an application accesses a large file. In the case of large file access, the NAS server, instead of returning the file to the NAS client, returns the location of the file instead. The NAS client then accesses the large file directly over a SAN, avoiding the overhead of going through the NAS server. This mode of access has some similarity to the way ST works; however, there are many important differences between ST and NAS accelerators.

Since NAS accelerators are like NAS in many respects, they, too, do not provide local file system semantics. Rather, they provide NAS semantics. As we have argued before, this may be inadequate for some applications that do data sharing. However, they are better than NAS in performance for large files because they allow direct SAN access for large files.

NAS accelerators are focused on improving large file performance. Since they do not use an improved protocol such as ST, they typically cannot improve the performance of access to small files in the way ST can. They also do not share many of the other goals of ST, such as improved storage management for heterogeneous computers, consistent data sharing, and massive scalability. They share many of the disadvantages of NAS, such as the lack of a global namespace and the need for remounting NAS servers when new servers are added.

Comparison to cluster file systems. Cluster file systems, such as GPFS[1] and xFS from the Berkeley NOW project,[24] are an alternative form of shared file system technology. Such file systems do not use a separate meta-data server. Cluster file systems differ from ST in several key ways. First, today's cluster file systems are designed to work only in homogenous server environments. That is, they will work if all servers are running the same operating system, such as AIX or Linux, but not in heterogeneous environments. ST, as we have discussed, supports heterogeneous environments. Although cluster file systems can probably be made to support heterogeneous server environments, it is much harder for them than it is for ST because the code to implement a cluster file system is larger and more complex than the code to implement an ST client. This makes the cluster file system harder to port and maintain on a diverse set of platforms. Second, cluster file systems run on tightly coupled clusters, where each computer knows the status of other computers in the cluster, and they all share a common mission or application. ST clients run on independent computer systems, which may or may not share a common mission. It is simpler to add another ST client than it is to add another computer in a cluster file system because ST clients run on independent computer systems. Cluster file systems are limited in scalability by the protocols used to synchronize meta-data and data updates. We expect the ST architecture to permit higher scalability because the clients are not synchronized, and because the ST servers operate on different parts of the namespace and also need no synchronization in normal operation. Finally, improving storage manageability is not a goal of cluster file systems, whereas it is a goal of ST.

Comparison to multiplatform file systems with separate meta-data servers. Finally, we need to compare ST to other multiplatform SAN file systems, such as the SGI Clustered XFS** (CXFS) file system,[25] that also use a separate meta-data server. CXFS has many similarities to ST; however, we believe that ST is better in the following respects:

- The ST meta-data servers support finer-grained load balancing. ST has the notion of containers, which are smaller than a file system, and each meta-data server handles all traffic from a container. On a meta-data server failure, the load on that meta-data server can be distributed to the remaining meta-data servers. In CXFS, one meta-data server handles one file system. When a meta-data server fails, it fails over to another meta-data server but does not leverage all of the other meta-data servers.
- ST supports policy-based storage management; CXFS has no such support.
- CXFS clients are part of a cluster; ST clients are independent computers. As a result, adding or dropping new clients is simpler in ST. When a new client is added in CXFS, all other clients need to agree that a new client is now a part of the cluster. No such agreement is needed in ST.
- Although CXFS has Windows support, it does not support NT lock modes, NT Access Control Lists, or case-insensitive names in the way ST does.
- CXFS meta-data servers are limited to the IRIX** operating system. ST meta-data servers are operating system independent.

Comparison to NASD. The Network Attached Secure Disk (NASD) project at Carnegie Mellon University[26] was among the first to propose that file servers should be removed from the data access path to achieve improved performance. Like ST, there are clients and there are servers that are mainly used for meta-data processing. For storage, they used a new kind of storage device (called object-based storage and discussed later), different from today's traditional block-oriented storage device. Rather than support commands such as read a block and write a block, their new storage device (NASD) supported commands such as read an object at offset X, write an object at offset Y. Furthermore, their NASD device supported enhanced security checking. This project implemented a version of NFS and AFS based on the use of NASD. Like ST, the clients make direct

access to the disk systems, and the server is used mainly for meta-data processing.

The NASD project was focused on data sharing between client computers rather than between servers in a machine room. In this way it was different from ST. Furthermore, ST goals for scalability and availability are more ambitious than the goals in the NASD project. Third, NASD did not provide policy-based storage management as ST does. Fourth, NASD clients were really NAS clients, so they provided NAS semantics, which we argue is inadequate for some types of applications that do data sharing. Finally, ST does not require the use of a new kind of storage device. As discussed in "Future directions for Storage Tank," ST has been designed to use object-based storage devices when they become available, but it does not require them.

Summary of comparisons with other approaches. In summary, ST is superior to NAS in performance, scalability, and consistency semantics. It is superior to DAFS in some aspects of performance and in compatibility with existing storage networks. It is superior to NAS accelerators in consistency semantics, scalability, and manageability. It is superior to cluster file systems in heterogeneous system support, manageability, and scalability. It is superior to NASD in consistency semantics, policy-based management, and use of unmodified storage systems. Finally, it is superior to other multiplatform SAN file systems such as CXFS in policy-based storage management and fine-grained server load balancing.

Future directions for Storage Tank

One aspect of the ST architecture that makes it so appealing is its flexibility. Although initially envisioned as a SAN-based file system and management solution, ST concepts lend themselves to solutions that go well beyond that model. Some of the areas of research that are currently being explored or envisioned include:

- Object-based storage devices: Storage devices that have knowledge of the objects (essentially files) being stored can add significant capability to a storage environment. For instance, the device can store specific keys with an object so that host systems that want to read or write the associated file are required to present the proper key before being allowed access to the object. Such an approach eliminates SAN security problems and makes it feasible to extend ST beyond the machine room to become a campus-wide file system. Additionally, object storage devices can conceivably optimize data placement (having intimate knowledge of drive geometry) or participate in intelligent caching or remote copy operations that are beyond the ability of current storage devices. Though not initially implemented, ST was developed with the intent to use object-based storage devices, and the current research prototype supports object-based storage devices.
- Geographically Distributed Tank: Although a campus-wide, SAN-based file system offers users and administrators many benefits, there are times when wide-area file access and sharing are required. As storage networks develop to cover larger areas (for example, through iSCSI technology), ST can be extended to provide the same single namespace and file sharing semantics over these wider distances. In this project, we are developing a new protocol for communication between ST server clusters. A request to access a file goes to the local ST server cluster. If the file is not on the local SAN, the request is forwarded to a remote ST server cluster on a remote SAN, and the file is then retrieved and stored on the local SAN before the local ST server cluster returns the block addresses of the local copy of the file to the ST client.
- zSeries* integration: Though originally implemented as an open systems solution, the architecture of ST is independent of any specific hardware or operating system. zSeries integration is a natural extension of the ST platform coverage. Although zLinux integration with ST should be relatively straightforward (given proper hardware capabilities), z/OS integration offers both potentially greater advantages and significantly more technical challenges. If successful, ST could be the first system to truly integrate open systems and z/OS file and data set access.
- Integration with autonomic storage systems: A truly autonomic storage system must include an overall autonomic manager that works cooperatively with all of the components of the storage system, including the file system, the storage devices and controllers, and the storage network infrastructure. Although it is not possible for ST (or any other file system) to implement a complete autonomic storage system on its own, the capabilities of the system can be extended to work more seamlessly in a fully autonomic environment. ST may well provide the model for future file system integration in an autonomic system.

Although these areas are by no means the only areas of future exploration envisioned for ST, they represent an interesting set of new directions for the technology.

Summary

There is a need for a storage system that fully exploits the benefits of storage area networks. Storage Tank is a powerful, comprehensive storage management solution that takes advantage of SAN technology and can address a wide range of customer requirements. It operates in a heterogeneous environment, which allows data sharing across many diverse platforms; provides massive scalability of data, servers, and clients; ensures high availability of customer data; and includes centralized, automated storage and data management that helps to reduce storage management costs. In addition, ST provides I/O performance comparable to that of local file systems and better than that of NAS systems.

ST continues to evolve. In the future, NAS gateways running on top of ST can take advantage of its ease of scalability and policy-based storage management; heterogeneous data sharing across wide-area networks will be possible; and ST can be integrated more seamlessly into fully autonomic storage systems.

Acknowledgments

We wish to acknowledge many people who have contributed to this project. Jai Menon, Bob Rees, and David Pease originally conceived the project, which took its present form in 1999. Bob Rees has been the chief architect of the project, and David Pease has been its manager. Randal Burns provided key architectural support for the project and completed his Ph.D. thesis based on Storage Tank. The key architects, designers, and implementers on the project from Research are Bob Rees, Wayne Hineman, Darrell Long, Demyn Plantenberg, Rajagopal Ananthanarayanan, Ralph Becker-Szendy, Bryan Henderson, and Miriam Sivan-Zimet. Cindy Sullivan provided technical writing support. The project was supported by the IBM Storage Systems Group (SSG) for the last two years. Linda Duyanovich managed the team from SSG San Jose. Bruce Hillsberg provided directional support from SSG. The project has been transferred to SSG Beaverton, where Tom Clark is the architectural leader. Gordon Arnold provides key customer insights and prioritizes future directions. We also wish to acknowledge Jonathan Haswell, Marc Eshel, Manoj Naik, Lance Russell, Edward Chron, James Myers, and Juan Gomez for their work on NAS over Storage Tank. Jonathan Haswell is the manager of this project. For their work on Distributed Storage Tank, we wish to acknowledge Leo Luan, Ted Anderson, Jeff Riegel, Manuel Pereira III, Binny Gill, and Chung-Hao Tan. Leo Luan is the manager of the Distributed Storage Tank project. In addition, we wish to acknowledge Norman Pass, the second-line manager for the Storage Tank, NAS over Storage Tank, and Distributed Storage Tank groups.

Cited references

1. F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," *Proceedings of the Conference on File and Storage Technologies (FAST)*, USENIX (January 2002), pp. 231–244.
2. *Autonomic Computing*, IBM Corporation, http://www.research.ibm.com/autonomic/.
3. Gartner, Inc., Stamford, CT, http://www3.gartner.com/.
4. IDC, Framingham, MA, http://www.idcresearch.com.
5. A. F. Benner, *Fibre Channel: Gigabit Communications and I/O for Computer Networks*, McGraw-Hill, Inc., New York (1996).
6. J. Menon and C. Fuentes, *iSCSI Performance and Architecture and Comparison to Other Protocols*, IBM Corporation, Almaden Research Center Report (February 21, 2000).
7. *InfiniBand Architecture Specification*, Volume 1.0, Release 1.0. InfiniBand Trade Association (October 24, 2000), http://www.infinibandta.org.
8. R. Weber, Editor, *SCSI Architecture Model-2 (SAM-2)*, Rev. 14, T10 working draft, International Committee for Information Technology Standards, Technical Committee T10 (September 2000), http://www.t10.org/index.html.
9. IBM TotalStorage Enterprise Storage Server, IBM Corporation, http://www-1.ibm.com/servers/storage/disk/ess/.
10. R. Nagar, *Windows NT File System Internals: A Developer's Guide*, O'Reilly and Associates, Cambridge, MA (September 1997).
11. A. Chang, M. F. Mergen, R. K. Rader, J. A. Roberts, and S. L. Porter, "Evolution of Storage Facilities in AIX Version 3 for RISC System/6000 Processors," *IBM Journal of Research and Development* **34**, No. 1, 105–110 (January 1990).
12. R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network FileSystem," *Proceedings of the Summer USENIX Technical Conference*, Portland, OR (Summer 1985), pp. 119–130.
13. P. J. Leach, *A Common Internet File System (CIFS/1.0) Protocol*, Technical Report, Network Working Group, Internet Engineering Task Force (December 1997).
14. J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebothom, and M. J. Wes, "Scale

and Performance in a Distributed File System," *ACM Transactions on Computer Systems* **6**, No. 1, 51–81 (February 1988).
15. J. P. Gelb, "System-Managed Storage," *IBM Systems Journal* **28**, No. 1, 77–103 (1989).
16. L.-F. Cabrera and D. D. E. Long, "Using Distributed Disk Striping to Provide High (I/O) Data Rates," *Computing Systems* **4**, No. 4, 405–436 (1991).
17. R. Becker-Szendy, *The Storage Tank Protocol*, IBM Corporation, Almaden Research Center Report (March 2002).
18. R. C. Burns, R. M. Rees, and D. D. E. Long, "Semi-Preemptible Locks for a Distributed File System," *Proceedings of the 2000 IEEE International Performance, Computing, and Communication Conference (IPCCC)*, (February 2000), pp. 397–404.
19. R. C. Burns, R. M. Rees, and D. D. E. Long, "An Analytical Study of Opportunistic Lease Renewal," *Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS)*, (2001), pp. 146–153.
20. C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," *ACM Transactions on Database Systems* **17**, No. 1, 94–162 (1992).
21. DB2 Data Links Manager, IBM Corporation, http://www.ibm.com/software/data/db2/datalinks/index.html.
22. D. Hitz, J. Lau, and M. Malcom, "File System Design for an NFS File Server Appliance," *Proceedings of the Winter 1994 USENIX Technical Conference*, San Francisco, CA (January 1994), pp. 235–246.
23. K. Magoutis, S. Addetia, A. Fedorova, M. I. Seltzer, J. S. Chase, A. Gallatin, R. Kisley, R. Wickremesinghe, and E. Gabber, "Structure and Performance of the Direct Access File System," *Proceedings of USENIX Annual Technical Conference*, Monterey, CA (June 9–14, 2002), pp. 1–14.
24. T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Y. Wang, "Serverless Network File Systems," *ACM Transactions on Computer Systems* **14**, No. 1, 41–79 (February 1996).
25. CXFS: An Ultrafast, Truly Shared Filesystem for SANs, SGI (Silicon Graphics, Inc.), Mountain View, CA, http://www.sgi.com/products/storage/cxfs.html.
26. G. A. Gibson, D. F. Nagle, W. Courtright II, N. Lanza, P. Mazaitis, M. Unangst, and J. Zelenka, "NASD Scalable Storage Systems," *Proceedings of 1999 USENIX Annual Technical Conference*, Extreme Linux Workshop, Monterey, CA (June 9–11, 1999).

Accepted for publication December 20, 2002.

Jai Menon *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (menonjm@almaden.ibm.com).* Dr. Menon is the department group manager of the Computer Science Storage Systems Group in IBM Research and Codirector of the Storage Systems Institute. He received a Bachelor of Technology degree in electrical engineering from the Indian Institute of Technology in 1977. He earned Master of Science and Ph.D. degrees in computer science from Ohio State University in 1978 and 1981, respectively. In 1982, he joined IBM Research in San Jose, California, where he became a pioneering researcher and designer of data storage systems and RAID (redundant array of independent disks) architectures. In May 2001, he was named an IBM Fellow. An IEEE Fellow, an IBM Master Inventor, and member of the IBM Academy of Technology, Dr. Menon has received numerous IBM technical awards, published 31 refereed papers and 46 technical reports, and is a contributing author to three books on database and storage systems. He also holds 42 U.S. patents and has 20 additional patent applications on file. He received the 2002 W. Wallace McDowell Award from the IEEE for "Leading Contributions to the Architecture and Design of Data Storage Systems and RAID Technology."

David A. Pease *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (pease@almaden.ibm.com).* Mr. Pease is a Senior Technical Staff Member in the Almaden Research Center and manager of the Storage Software department. Since 1996, he has managed the Storage Tank research project. He began working on storage systems research projects at Almaden in 1990. In addition to Storage Tank, he has worked on projects relating to the Tivoli Storage Manager product (formerly known as IBM ADSM) and the Universal Disk Format (UDF) file system for DVDs. For 12 years prior to joining IBM, Mr. Pease ran his own business, consulting and teaching software development and operating systems. In 2000, he completed his work for a Master of Science degree in computer engineering at the University of California, Santa Cruz, where he is currently a Ph.D. degree candidate.

Robert (Bob) Rees *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (rees@almaden.ibm.com).* Mr. Rees is a Senior Technical Staff Member at the Almaden Research Center. He joined IBM Research in 1982 and has received several IBM technical and Invention Achievement awards for his work in storage-related systems. Most notably, he was the chief architect and team leader of the Tivoli Storage Manager product (formerly known as IBM ADSM) and is currently the chief architect and team leader for the Storage Tank SAN file system project. Mr. Rees received a B.S. degree in computer engineering from the University of California, Santa Cruz.

Linda Duyanovich *IBM Systems Group, 5600 Cottle Road, San Jose, California 95193 (lduyanov@us.ibm.com).* Ms. Duyanovich is a senior program manager for IBM's Storage Software Products. From 1999 to 2002, she built and managed the product development team that worked jointly with IBM Research in Almaden on the Storage Tank project, targeted at the storage network environment. Over her more than 20 years with IBM, Ms. Duyanovich has held a variety of management and technical positions in storage systems development and performance. From 1996 to 1999, she was Vice President of Product Development at MatriDigm Corporation, where she led development of rules-based software maintenance tools. She received a Bachelor of Science degree in mathematics from Stanislaus State College in 1976 and earned a Master of Science degree in operations research from Stanford University in 1977.

Bruce Hillsberg *IBM Systems Group, 5600 Cottle Road, San Jose, California 95193 (blh@us.ibm.com).* Mr. Hillsberg is Director of Storage Software Strategy and Technology for the Systems Group. He is responsible for developing the technology strategy for IBM storage infrastructure software, identifying and incubating new technologies, and identifying emerging opportunities for the storage software business. He joined IBM in 1981 and has spent most of his career managing the development of networking and systems management software. Most recently, Mr. Hillsberg has worked in the IBM Software Group on middleware integration initiatives. Previously, in IBM's Banking, Finance, and Securities Industry Solution unit, he was responsible for the Year 2000 testing strategy for IBM's internal applications, and he developed Internet appliances for the small business marketplace.