UCRL-PROC-208872

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

High Performance Storage System Scalability: Architecture, Implementation, and Experience

R. W. Watson

January 6, 2005

22nd IEEE 13th NASA Goddard Conference on Mass Storage Systems and Technologies
Monterey, CA, United States
April 11, 2005 through April 14, 2005

High Performance Storage System Scalability: Architecture, Implementation and Experience

Richard W. Watson
Lawrence Livermore National Laboratory
dwatson@llnl.gov

Abstract

The High Performance Storage System (HPSS) provides scalable hierarchical storage management (HSM), archive, and file system services. Its design, implementation and current dominant use are focused on HSM and archive services. It is also a general-purpose, global, shared, parallel file system, potentially useful in other application domains. When HPSS design and implementation began over a decade ago, scientific computing power and storage capabilities at a site, such as a DOE national laboratory, was measured in a few 10s of gigaops, data archived in HSMs in a few 10s of terabytes at most, data throughput rates to an HSM in a few megabytes/s, and daily throughput with the HSM in a few gigabytes/day. At that time, the DOE national laboratories and IBM HPSS design team recognized that we were headed for a data storage explosion driven by computing power rising to teraops/petaops requiring data stored in HSMs to rise to petabytes and beyond, data transfer rates with the HSM to rise to gigabytes/s and higher, and daily throughput with a HSM in 10s of terabytes/day. This paper discusses HPSS architectural, implementation and deployment experiences that contributed to its success in meeting the above orders of magnitude scaling targets. We also discuss areas that need additional attention as we continue significant scaling into the future.

1. Introduction

The High Performance Storage System (HPSS) provides scalable hierarchical storage management (HSM), archive, and file system services. Its design, implementation and current dominant use are focused on HSM and archive services. It is also a general-purpose, global, shared, parallel file system, potentially useful in other application domains. When HPSS design and implementation began over a decade ago, scientific computing power and storage capabilities at a site, such as a DOE national laboratory, was measured in a few 10s of gigaops, data archived in HSMs in a few 10s of terabytes at most, data throughput rates to an HSM in a few megabytes/s, and daily throughput with the HSM in a few gigabytes/day. At that time, the DOE national laboratory[1] and IBM HPSS design team recognized that we were headed for a data storage explosion driven by computing power rising to teraops/petaops requiring data stored in HSMs to rise to petabytes and beyond, data transfer rates with the HSM to rise to gigabytes/s and higher, and daily throughput with a HSM in 10s of terabytes/day. Therefore, we set out to design and deploy a system that would scale and evolve from the base above toward these expected targets. These targets have been successfully met.

While the rapid increase in both computational power and memory, storage device capacity, and networking bandwidth have made these increases in storage system capacity and performance possible, without proper attention to software architecture, implementation and deployment, this hardware potential can not be fully realized or exploited. Even assuming new faster hardware and a properly designed and implemented storage system, successful scaling, particularly for data transfer, is not just a matter of plugging in the new hardware, changing a few configuration settings and running the system. It requires careful attention to all phases of the end-to-end process.

There are many dimensions of scalability to which a storage system architecture and implementation must pay attention. This paper discusses those dimensions and illustrates the architectural approach and some of the implementation choices and deployment experiences that have facilitated achieving scalability in these dimensions. It also discusses some areas where further work is

[1] Lawrence Livermore (LLNL), Los Alamos (LANL), Lawrence Berkeley - National Energy Research Supercomputer Center (NERSC), Oak Ridge (ORNL), and Sandia (SNL) National Laboratories.

required as the system continues to scale across these dimensions in the future.

Scalable data throughput: This dimension focuses on end-to-end I/O throughput, for both single files and for the aggregate throughput of many simultaneous file transfers or I/O operations.

Scalable storage capacity and storage space management: This dimension includes scaling storage capacity, numbers and types of storage devices, and files and file sizes. It also includes scalable space management for migration and purge of disk cache.

Scalable robustness: This dimension includes the ability of the system to (1) tolerate or recover from hardware failures without loss of user data or system metadata and (2) to maintain the consistency of both user data and system metadata in the face of concurrent accesses during normal operation.

Scalable name service: This dimension for HPSS involves a scalable hierarchical directory service with virtually unlimited numbers of directories and directory entries, and a global name space spanning multiple distributed HPSS systems. It also includes scaling the number of simultaneous directory accesses and access performance.

Scalable numbers of clients: This dimension includes both increasing numbers of end users and internal clients and associated concurrent operations.

Scalable deployment across geographical distances and multiple cooperating institutions: This dimension involves distribution of data storage devices and metadata for performance and robustness, and integration of multiple storage systems into a global namespace and secure environment.

Scalable storage system management: This dimension enables system administrators to manage and configure hundreds of devices at a time with ease and convenience, obtain tuning information and set tuning parameters, monitor system health, perform diagnostic operations in a complex environment, and support rapid creation and modification of management screens by developers as the system and environment evolve.

Scalable security: This dimension involves a security infrastructure that supports the scaling and distribution of users, servers and associated devices across networks and multiple sites.

Client roles in scalability: This dimension includes the role of storage utility clients or agents in supporting scalability (e.g. client applications for use in data storage and retrieval that help achieve optimum data transfers and other uses of the system).

2. HPSS high-level architecture

Figure 1 shows a high-level view of the HPSS architecture, which was guided by the IEEE Mass Storage System Reference Model (MSSRM) V5 [19]. There are two key features of this architecture central to HPSS's scalability success. First, there is its network-centric architecture, where its Metadata Service is out-of-band with data transfer. (Metadata is that information used externally and internally to name, protect, and locate the system's storage objects, such as virtual volumes and files). Data sources and sinks (e.g. storage devices, memory, file systems) are either virtualized by the Movers to create real or virtual network-attached storage devices that can communicate directly over a network (e.g. TCP/IP networks such as gigabit Ethernet (GigE), or a Storage Area Network (SAN)) or Movers can make devices directly accessible from the client systems on a SAN [9].

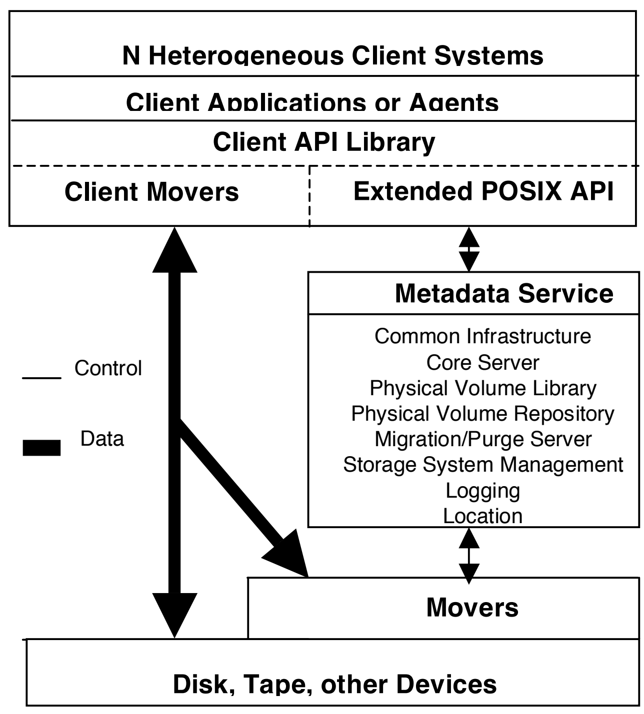

Figure 1. High-level HPSS architecture

Thus, to scale system throughput one adds more network bandwidth and more Movers with their virtualized storage devices or one adds more directly attached SAN devices. The HPSS architecture also supports "third-party" transfers where an application (the third-party) can issue I/O operations that cause the data to go directly between sources and sinks on two other systems thus, avoiding a trip through the third-party (copying) application's memory.

Second, the robust Metadata Service, with its separate associated metadata storage, is cleanly modularized supporting well-defined scalable numbers of abstract and physical storage objects. The architecture shown in Figure 1, where the metadata services are out-of-band with the data path is common in SAN and clustered global file systems [12,13,17,22,26,31]. This centralized, asymmetrical metadata architecture is used by HPSS and other systems for three main reasons. It simplifies: (1) lock management for concurrent accesses to shared user data, (2) metadata integrity, consistency and recovery because the metadata is controlled by a single component, simplifying the algorithms and enabling use of a robust commercial RDBMS as the metadata engine, and (3) securing the metadata by placing it on a separate system from one running user programs or directly accessible by such systems, thus minimizing the components that have to be trusted. In particular, there is increased security because all accesses can only take place through an authenticated message interface and because having no user code running on the metadata system minimizes the ability of an unauthorized person hacking into the system and compromising the entire metadata system and thus all the data managed by the storage system.

Managing all metadata in a central metadata system makes it easier to support HSM and archive services for many, generally changing, heterogeneous file systems, because there is no sharing of metadata between the file systems and HPSS. Assuring high metadata robustness and security is important to the HPSS design because HPSS operates in environments where metadata must be safeguarded over decades and all client systems and their users cannot be completely trusted.

There are two main issues with a centralized Metadata Service. One, since a central Metadata Service is a single point of failure, redundant metadata computers and disks may be required, with either manual or automatic failover mechanisms or procedures for switching between them. Two, there is some extra operation latency in systems using a central Metadata Service. This results because all requests and replies go through the network control path, usually a LAN, to the Metadata Service. This is the case for both relatively infrequent pure metadata operations like "create" and "rename" and, also the more frequent initiation of the data transfer operations "read" and "write", even though the actual data transfers take place concurrently directly between client and storage system on separate network connections. I/O throughput scaling proceeds smoothly, even with some extra operation latency, when the average data transfer time for "reads" and "writes" is "large" relative to the time required to perform the metadata functions and control communications for each I/O operation. This is generally the case for HPSS deployments, where the bulk of the data being transferred is in relatively large files. These issues are discussed further later.

Below brief outlines are given of the function for the components shown in Figure 1. Much more detail is in [5].

Data Transfer Components

Mover: A Mover either directly transfers data from a source device to a sink device or can redirect control of the I/O to another Mover (e.g. for a direct SAN transfer [9]). A device can be a memory, network, tape, disk or file system or other physical or logical storage entity. The Mover's client (typically the Storage Service or a client application) provides I/O descriptors that describe the location of the data to be transferred. It is the Mover's responsibility to transfer the data, retry failed requests, and attempt to optimize transfers. (Note, in SAN operation the SAN-aware Mover on the HPSS side is not directly involved in the I/O operation as it passes metadata (an I/O descriptor) to the Client Mover or client application for direct access to the data [9]). When the Mover virtualizes devices into network-attached storage devices, they can be used securely on a TCP/IP network because the Mover will only accept a data transfer command from an authenticated client such as the Storage Service. The Mover is a simple, versatile, modular, component supporting easy evolution to new types of networks, storage devices and storage entities such as files.

Metadata Components

Core Server

Three services, Bit File Service, Storage Service, and Name Service, interact extensively. We recently integrated these three services into a single process named the "Core Server", in Figure 1, reducing communications and nested atomic transaction overhead between them.

Name Service: The Name Service provides a Portable Operating System Interface (POSIX [20]) view of the global hierarchical name space, translating a human-oriented name to an HPSS unique-object-identifier. The name spaces of different HPSS systems can be joined to create a larger distributed shared naming environment.

Bitfile Server: The Bit File Service provides the abstraction of logical bitfiles (the term "bitfile" is IEEE MSSRM terminology, even though files are logically streams of bytes) to client applications. Clients may reference random portions of a file. The POSIX file API has been extended to support parallel reading and writing of file data [5].

Storage Service: The Storage Service provides a virtual volume service supporting a hierarchy of storage objects such as storage segments and virtual volumes (e.g. striped, mirrored, multi-physical volume). The Storage Service schedules the mounting and dismounting of removable media through the Physical Volume Library and provides the Movers with I/O descriptors to perform the actual I/O.

Migration/Purge Server: The Migration/Purge Server manages the placement of data on appropriate storage media within storage hierarchies using site specified policies.

Storage System Management: Storage System Management provides both a GUI and command line interface enabling system administrators to configure, monitor and control HPSS resources. All other system components provide information to a system management metadata base and provide other notifications and alarms.

Location Server: The Location Server enables its clients to locate servers and gather information from both local and remote HPSS systems.

Logging Service: Any HPSS component can send messages to the Logging Service to record events used for security auditing, problem troubleshooting, tuning and other needs that might develop.

Components with both data transfer and metadata functions

Physical Volume Library: The Physical Volume Library manages all HPSS physical volumes. It is in charge of coordinating the mounting and dismounting of sets of physical volumes, and allocating drive and cartridge resources to satisfy mount and dismount requests. The Physical Volume Library supports atomic mounts of sets of cartridges for parallel access to data on striped virtual tape volumes [3].

Physical Volume Repository: The Physical Volume Repository coordinates the mounting and dismounting of cartridges and performs other cartridge related operations. Each Physical Volume Repository is typically configured to manage the cartridges for each robot complex utilized by HPSS.

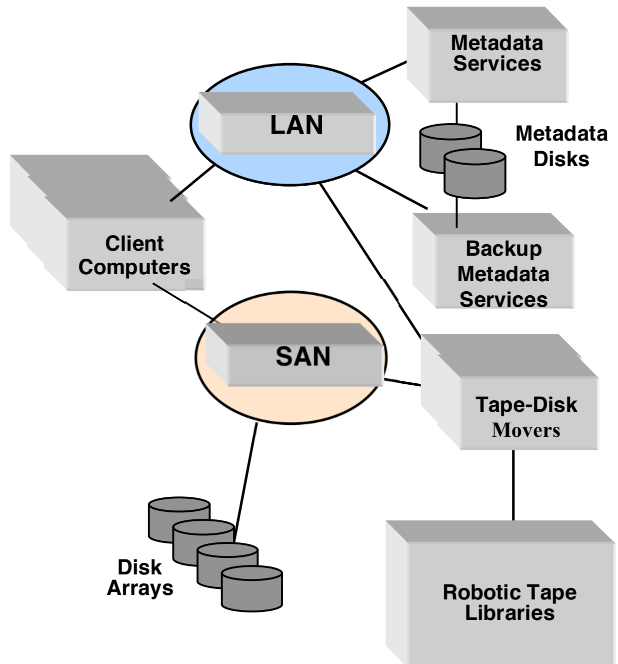

Figure 2. Abstracted HPSS Deployment

HPSS functionality, infrastructure and implementation are constantly evolving. Each HPSS component is implemented as a separate multithreaded process that communicates with other HPSS components by remote procedure calls. This modular, distributable implementation, along with the separation of Metadata Service from data transfer, has been very successful. The component modularity has also proved very useful as part of the distributed development team's development methodology.

3. HPSS second level architecture, implementation and experience issues related to scalability

3.1. HPSS infrastructure

A key second level architecture and implementation decision was to define a set of common infrastructure services required by HPSS servers and clients. These infrastructure services (called "common infrastructure" in Figure 1) provide a uniform implementation foundation and serve to "glue together" the distributed clients and servers. All HPSS components must work together to provide users with a stable, robust, secure, distributable and portable storage system. Separation of the infrastructure simplifies development by letting HPSS developers focus on the storage application, increases robustness (when well tested and widely used commercial components are used), and supports evolution as industry standards and products evolve in functionality, performance and support. The common infrastructure organizes the interfaces to the HPSS clients and servers into four primary services: remote procedure calls (RPCs), metadata system, security services, and concurrency services (threads).

We initially chose the Distributed Computing Environment (DCE)[34] as a central infrastructure component because it provided an integrated set of RPC, security, thread, and time services. We chose IBM's Encina/Structured File System (SFS) (built on DCE) yielding a well-integrated distributable infrastructure [28]. After a decade of service, we recently replaced Encina/SFS with IBM's DB2 Universal Database [18] in HPSS Release 5.1 and DCE with a set of industry standard security, RPC and thread services in Release 6.1. These decisions were made to improve performance and robustness, and because IBM announced a phase-out of Encina/SFS and DCE support.

3.2. Metadata system

Fundamental to both HPSS robustness and scalability is the metadata architecture and implementation: a central Metadata Service, a robust metadata engine supporting atomic transactions, a scalable set of metadata data structures, scalable update and access algorithms, backup and recovery mechanisms, and separation of metadata from user data storage. While robustness is important to any file system, it is particularly critical to HSM and archive applications, where data is intended to live possibly for decades. If the metadata is damaged or lost during metadata updates, then user data, while correctly stored, will not be accessible or will be corrupted.

We initially chose IBM's Encina transaction manager with its associated Structured File System (SFS) because it supported distributed, nested atomic transactions, record structures organized by multiple keys, and it was integrated with DCE [7,28]. SFS, at the time, was also faster than RDBMS's running on the available minicomputer machines targeted for running the Metadata Service components and had other capabilities not common in RDBMS's available in 1992.

To improve metadata scalability and performance, and to add new levels of robustness, we recently examined the options available for metadata engines. We focused particularly on the commercial RDBMS systems most widely used in business and mission critical applications, and thus having demonstrated high robustness. Benchmarking showed that their performance and capabilities had improved dramatically since HPSS was initially implemented. In fact these tests indicated that we could expect to see a significant metadata performance improvement over Encina/SFS. Further, these systems were expected to be even more robust because they were much more widely deployed in business and mission critical applications, and thus had higher levels of support. Because of the richness of their functionality, they also offer the potential for metadata optimizations that can yield higher performance as the HPSS implementation evolves.

We chose, as mentioned above, IBM's DB2 Universal Database as our current RDBMS metadata engine [18]. Our implementation isolates the RDBMS choice from its HPSS client processes through a metadata manager library. This layer provides HPSS components with a means of accessing metadata, while insulating other parts of the system from the underlying details of the metadata RDBMS engine. This will also simplify the process of switching to a different RDBMS in the future, should this prove desirable.

Metadata robustness requires a well engineered and tested metadata engine and a set of mechanisms to tolerate or recover from hardware failures. HPSS supports several levels of metadata integrity protection. First, is the use of a commercial RDBMS that is thoroughly tested, widely used, highly tuned and well supported. Second, the metadata engine supports atomic transactions, to assure metadata consistency in the face of possible failures when in the middle of a sequence of metadata updates that must all occur correctly or not at all [7]. Third, the metadata is stored in a mirrored RAID disk system, with recovery automatically handled by the RAID controller or the operating system. Fourth, the metadata disks are backed up daily. Last, an alternate failover machine, either inserted automatically [16] or manually, is available to run the Metadata Service. Cleanly separating metadata storage from user data storage makes it possible to keep all the metadata on high performance stable storage, and perform fast metadata restore operations, should they be necessary, time independent of the amount of user data stored. This is an important consideration in a system storing petabytes of data.

3.3. Scalable data structures and algorithms

Careful attention is required in the design of metadata structures and manipulation algorithms to enable scalability across the dimensions above, particularly scalable capacity: total storage space, as well as numbers and sizes of objects such as files and directories. It is beyond the scope of this paper to discuss this topic in detail, but a few remarks are in order. One of our experiences building earlier storage systems was that decisions on field lengths in metadata records might limit scalability in numbers of objects or object sizes. Thus, from the beginning, HPSS has used 64 bit field sizes and 64 bit arithmetic widely, particularly for numbers and sizes of objects, to avoid artificial field size restrictions. Another restriction limiting scalability of systems has been metadata storage limitations and requirements to statically allocate fixed disk regions for metadata use, such as inodes. HPSS has no such limits and makes effective use of the large and dynamic storage capabilities of DB2 for table and index growth.

Keeping track of free disk space and allocating space are central storage system functions. To support scalable capacity and I/O one wants to allocate space in as large as possible contiguous regions, rather than fixed size blocks. There are two central internal objects in HPSS associated with these functions, the storage segment, out which files are constructed, and the virtual volume block, that can be thought of as the stripe stride length. Disk segments are variable length, within parameters set by a system administrator, and map to contiguous regions on disk. Tape segments can be any length up to the length of a tape. Segments are safely kept track of in dynamic tables within DB2. HPSS keeps track of disk free space in tables kept in memory, which can be quickly rebuilt from the segment metadata if needed. Further, HPSS metadata manipulation uses the efficient B+ tree and other mechanisms provided by DB2, and in the case of algorithms outside of DB2, like the disk free space mechanism above, care is taken to assure they will scale well for large numbers and sizes of objects.

3.4. Security services

Mechanisms are provided that allow HPSS components to communicate in an authenticated manner, to authorize access, to enforce access control (POSIX user, group, world and access-control-lists (ACLs)) on HPSS objects, and to issue log records for security-related events. Security policies are site dependent. HPSS provides a clean separation of policy and mechanism so that sites can use default security services or add both their own policy modules and security mechanisms, in a plug-and-play architecture. Industry standard services are used, such as the Generic Security Services (GSS) API, Kerberos, and directory services that support HPSS data schemas, (such as Lightweight Directory Access Protocol (LDAP) [23,24,35]. The scalability implications for the security services are primarily in the distributed, cross-security domain, including a multi-institution dimension, where DCE excelled because it provided a fully integrated set of services with an essentially uniform implementation across platforms on which HPSS was targeted. However, DCE did have the drawback that sites not using DCE on an institution-wide basis had to develop DCE expertise for use with HPSS. Further, we had to develop a non-DCE gateway to support access to HPSS from client machines that did not run DCE. Since today there is no industry-supported product with DCE's integrated set of services, we selected an equivalent set of industry standard services (i.e. ONC RPC [30], POSIX threads [21], the security services above) and integrated them as a DCE replacement. To provide cross-realm integration, we use the multi-realm capabilities of Kerberos and LDAP. Kerberos is used to authenticate users, servers, etc. and LDAP is used to store registry information associated with all users. Multiple LDAP servers at different sites can be integrated to provide a distributed registry service. We require LDAP servers to support strong authentication such as Kerberos and ACLs to securely control registry information.

3.5. Concurrency

Concurrency is critical to supporting and scaling multiple simultaneous operations throughout the system and being able to scale server performance, when needed, by multiprocessing HPSS components on commodity clusters of processors. IEEE POSIX-compliant threads and POSIX thread interfaces are used [21]. POSIX threads are supported as kernel-level threads by the operating system vendors. The central implementation architecture is that all HPSS components are implemented as multithreaded and thread safe processes. As each request arrives, a separate thread is forked to handle it. As commodity multiprocessors have arrived on the market, HPSS has been able to scale its metadata handling cost effectively. Organizing the metadata so that only minimum units of granulariy need locking is important for scalable concurreny.

3.6. Communication services

Communication (LAN, WAN, SAN) is central to the HPSS architecture and its I/O scalability. Therefore, providing mechanisms that support optimized use of each site's available communications capabilities has received close attention. There are several levels of communication protocols and mechanisms used within HPSS. The lowest level is network communications. Currently TCP/IP and use of TCP/IP networks such as GigE and 10 GigE are dominant. Early in HPSS history, HIPPI networks and the IPI3 protocol were supported, along with directly attached HIPPI devices from companies such as Maximum

Strategy [32]. GigE replaced HIPPI both in cost and reliability. To achieve full use of TCP/IP networking, we pay close attention to tuning networking parameters and resources such as packet and buffer sizes (see Section 4). Fibre Channel SAN has also emerged in recent years along with LAN, WAN, and SAN use of iSCSI. HPSS supports use of SAN networking [9]. SANs potentially improve performance of migration and caching between device hierarchy levels [9] and can be used for interconnecting HPSS storage devices. HPSS can take advantage of each site's networking configuration, which may include multiple networking technologies and separate different size data transfers on different network routes. As mentioned earlier, the modularization and virtualization of networking into the Mover component has been central in easily tracking the constantly evolving networking technologies.

Above TCP/IP, HPSS uses remote procedure call (RPC) protocols for its control communications between clients and servers and between internal server components. Control communication between the Storage Service and Movers uses simple socket communications because the generality of RPC communications is not required as only two types of messages are exchanged between them.

Another protocol in use within HPSS is the Mover-to-Mover data transfer protocol [6]. It was developed to provide a simple data transfer protocol layer within low latency LAN and system area networking environments and to support a variety of transfer mechanisms such as IPI-3 over HIPPI, shared memory, and TCP/IP sockets. It uses a message exchange flow and error control scheme for each block transferred. It also supports negotiations of transfer optimizations available between pairs of data sources and sinks, using information contained in a configuration file. Further, it supports striped transfers between different numbers of Movers on the client and the HPSS sides. It is simple and works well in low latency network environments. As HPSS usage scaled to WANs, message exchange latency increased, limiting bandwidth performance scaling. The Mover protocol evolved to utilize TCP/IP's streaming capabilities to eliminate these latency-scaling limitations.

At the next level, HPSS can scale bandwidth and throughput by different forms of network striping. HPSS client data transfer agents such as PFTP, HSI, PSI, and HTAR (see Section 3.10) can stripe transfers over multiple TCP/IP connections, utilize multiple network interface cards (NICs), when available, or utilize multiple nodes such as on Linux Clusters, or use combinations of the above [5,8,10]. In a cluster environment, the HPSS client agents may utilize internal networks, such as Quadrics or Myrinet for high-speed intra-cluster transfers. The number of TCP connections used is balanced against the performance of the NICs in use. Further, HPSS can do striped file transfers between M Movers on the client side and N Movers on the HPSS side, where these Movers can be on the same or different nodes of a system. (The HPSS file stripe width is N.) Connections between Movers can use multiple NICs and/or TCP connections. The architecture of Movers coupled with the Mover-to-Mover Parallel Transport Protocol (PTP) [6] makes this powerful generality and scaling capability possible.

Client agents (discussed in Section 3.10) add the final level of protocols and mechanisms to support high I/O throughput. Optimum usage of the above forms of communication striping requires proper set up of various configuration files (used by both Movers and the client agents), along with other parameters such as packet and buffer sizes.

3.7. Device striping

I/O scaling is at two levels, bandwidth to individual files and total aggregate I/O throughput. Besides striping of connections, NICs, Movers, and nodes as discussed above, software striping of devices at the HPSS level is also used for scalable data transfer. The Storage Service provides the striped virtual volume abstraction. Since RAID disks provide a virtual highly reliable disk, we can safely stripe RAID devices very widely in HPSS to achieve highly scalable disk transfer rates. Disk striping in production environments are currently rather modest with four-way stripes being the largest, thus, there is significant scaling headroom using wider stripe widths. Demonstrations using 16-way and even 256-way disk stripes have been performed from HPSS's earliest days.

HPSS can also support virtual striped tape volumes to improve transfer rates to tape. The Storage Service supports virtual volumes of Redundant Arrays of Independent Tapes (RAIT), Level 0 (mirroring) and Level 1 (striping with no parity tape). Current usage generally limits operational tape striping to four-way RAIT Level 1 stripes because of concerns about the reliability of tape media and drives, although other strategies for using wider stripes are being considered. For example, it is possible to write a file to tape directly and then make a deferred mirrored copy later on a scheduled basis. We had hoped to encourage industry to produce a commercial high performance RAIT Level 3 or 5 controller and to that end a U. S. DOE project was created to develop such a controller. A prototype was developed, but the contractor did not elect to move it to the product stage. DOE labs are prototyping two RAIT Level 3 or 5 approaches at relatively low priority, one using software as a RAIT controller that could be used within HPSS, and the other using a Linux box as a RAIT controller. Other tertiary storage technologies such as MAID disks [15] and holographic devices [25] may eventually become economically competitive with tape for use in large archives.

3.8. Storage hierarchies, classes-of-service and file families

HPSS supports the ability to organize classes of devices into varieties of storage hierarchies, from single-level hierarchies (e.g. disk for small files or tape for large files) to more complex hierarchies involving multiple levels of devices (e.g. a disk cache with multiple tape levels below it or disk over mirrored tape copies). Use of multiple storage hierarchies in a given deployment allows use of a range of media and devices with different costs and performance for different classes of service (COSs) and is central to HPSS's capacity and I/O scaling strategies. A COS is selected for each HPSS file and is defined based on characteristics such as file size, disk type and speed, stripe width, tape type, and whether tapes are mirrored [4].

Selection of the proper COS for a file is achieved via a "hints" mechanism or by a system administrator setting. For instance, small files requiring quicker access may best be stored only on redundant disk devices or might be best directed to dual-hub access-efficient tape media, as opposed to capacity-focused devices. An example of COSs and multiple hierarchies is the current LLNL environment where five main COSs are used: small files (<4MB), medium files (4MB – 32MB), large files (32 – 256 MB), jumbo/htar (> 256 MB), and dual-critical (large/jumbo files that are mirrored to tape). Each of these COSs uses different hierarchical sets of storage devices available within the LLNL system. The HSI application (see Section 3.10) augments this capability with a sophisticated auto-COS-selection capability that requires no user knowledge of the site's COSs, and allows sites to segregate resources by user, group, and/or account, and to choose between hierarchies based upon the desired number of copies, among its many features.

File Families are a way to assure files in a directory sub-tree are co-located on media. This is done when it is common to access multiple files from the same File Family during an application run, and thus, reduces the number of tape mounts and increases throughput. File Families and configurable COSs allow HPSS to scale to satisfy wide ranges of application requirements by matching classes of data with the best and most efficient device/media combinations. For example, climate research applications typically write extremely large files and so at some sites are configured to use a Direct-To-Tape class of service, with no disk in the hierarchy at all. Weather sampling applications, on the other hand, collect a large number of very small files, which are frequently accessed. These are therefore written to a COS with a large disk cache at the top of the hierarchy with an associated migration/purge policy that encourages lengthy retention of files at the top of the hierarchy.

Associated with storage hierarchies is the need to support file migration and purge [5]. Migration of a file down the hierarchy (to slower, less expensive media) is performed in order to free space on more valuable media, take advantage of lower cost or new types of media, or implicitly provide backup. Dynamically configurable site policies are used to determine under what conditions migration should take place and when purging should take place. There are two performance and robustness issues here. One, by setting an aggressive migration policy, but delaying purge (thus potentially speeding up read back performance), backup is automatically provided as part of the hierarchical services, without a special backup service being required as two copies exist, one on each level of the hierarchy. Two, when migrating to tape, mirrored copies (possibly distributed) can be created as part of the policy, further enhancing user data robustness.

Copies can also be explicitly (by command) or implicitly (when accessed) staged up the hierarchy for higher performance. A capability that is very important to scalability is the creation of metadata records that indicate which files need to be migrated and purged. Files that are written are marked as candidates for migration and those that have been migrated as candidates for purge. By doing this, HPSS eliminates the need to scan through large amounts of metadata in order to determine which files are candidates for migration and purge. Without this mechanism there would eventually be a scalability limit on the migration/purge algorithm. Sites also take advantage of the multiple mirrored copy feature of hierarchies to create backup copies in separate physical locations to protect against fire or other disaster.

A deployment issue related to a high performance tape environment is configuring the system so that there are multiple robotic arms available, either in single tape library or by using multiple libraries, so that multiple tape mount/dismount requests can be in progress at once. HPSS has logic to optimize the choice of a tape drive in multi-robot configurations.

3.9. Subsystems

HPSS can distribute Metadata Services across multiple Core Servers and databases using a feature called

"Subsystems" (a Subsystem can be thought of as a namespace partition along with its associated media, managed by its own Core Server). Subsystems differ from whole instances of HPSS in that a single name space is still maintained, a single login is used by client applications, and devices such as tape and disk drives can be accessed by all subsystems within a single HPSS. It simplifies system administration by maintaining a single system image. Subsystems can be placed on separate computers or in separate processes on shared computers as appropriate to the requirement.

An example use of Subsystems is the European Center for Medium Range Weather Forecasting (ECMWF), which uses Subsystems to separate two very different application domains within a single HPSS instance, an interesting form of scaling. ECMWF uses three Subsystems. Subsystem 1 contains the center-wide name space root of roots. Subsystem 2 is the MARS subsystem that receives data in near real-time from weather sensors around the world. MARS is a sophisticated data access application that uses HPSS as its backing store and is a tape-only system. Subsystem 3 is the general-purpose user file store, ECFS, which has large disk and tape layers.

The current HPSS implementation of Subsystems has the disadvantage that it partitions by name space sub-trees. With the current implementation and available tools, a system administrator is required to make the decision whether to add a new Subsystem at system set up, or when new users are added, and thus partition based on anticipated loads across the metadata system. To improve Subsystem dynamic configurability, additional load measurement capabilities and partitioning tools are required. A better long-term solution would be to support automated distribution of resource load across Subsystems. Subsystems do, however, provide a foundation for further evolution.

3.10. User interfaces

HPSS supports an extended POSIX Client API (CLAPI) as its most complete and powerful interface. The HPSS Client API is a superset of POSIX semantics, providing fields to support striping, classes of service, and other HPSS functions that POSIX semantics do not address [5]. The HPSS CLAPI can be used directly by end user applications or by data storage service applications (client agents) or with other interfaces. CLAPI supports a powerful list form of I/O transfer command that allows multiple operations to be initiated with a single command. Achieving optimum parallel I/O can be a complex problem, involving scalability issues in areas such as file system, network and I/O striping, debugging and troubleshooting, resource allocation, error recovery and job restart capability. Therefore, client agents are available with HPSS or may be written by sites that facilitate applications achieving maximum use of HPSS's I/O scaling capabilities. The MARS and ECFS applications at ECMWF, mentioned above, are examples of site agents. PFTP, discussed below, is an example of an included HPSS agent. Transparent access to the HPSS CLAPI through the Linux Virtual File System (VFS) capability is currently in its prototype phase.

Parallel File Transfer Protocol (PFTP) [5], Hierarchical Storage Interface (HSI) [10] and Parallel Storage Interface (PSI) [8] are three agents with the configuration "knowledge" and mechanisms to optimize system throughput and parallel network and I/O bandwidth. An example use of configuration knowledge to optimize throughput is the ability of both HSI and PSI to use HPSS's extended file attributes (indicating information such as file location - disk or tape and if on tape which tape and where on that tape). Thus, HSI and PSI can use this information to schedule file transfers to minimize tape mounts. Another example is the ability of PFTP and HSI to use HPSS configuration information to set up the optimum network-striping environment for a transfer. HSI can make use of HPSS's ability to support third-party transfers and set up such transfers between Movers on any pair of systems, even between Movers on separate HPSS systems of cooperating sites. HSI and PSI also support a range of other useful file and transfer functions.

Another utility developed to improve transfer performance is HPSS Tar (HTAR) [10]. HTAR creates POSIX-compliant TAR files directly in HPSS, using multithreading and HPSS striped transfers to achieve high I/O rates by blocking many small files into a single large archive file. HTAR includes a number of other features, such as automatic creation of a separate index file, that facilitate random retrievals of HPSS files and listing of archive (tar) files that may reside on tape. HTAR provides a simple means to scale the writing and reading of large numbers of small files in a single high data throughput transfer to HPSS.

HPSS supports The Open Group's XDSM interface [33], also known as DMAPI, which supports the connection of a file system to a tape archive system. HPSS supports an XDSM interface to the open source Linux XFS file system [29] and to IBM's Distributed File System (DFS) [15]. IBM is discontinuing DFS support, but it is still in use at many sites. Indiana University, for example, uses DFS with HPSS to support over 1300 users in two geographical campus locations. Since IBM support for DFS is being phased out, existing DFS/HPSS filesets will be converted to HPSS-only filesets. These filesets will then be made available to clients using either the

HPSS NFSv4 [27] under development by Commissariat à l'Energie Atomique/Division des Applications Militaires (CEA/DAM) Compute Center in France or the HPSS VFS Client implementation.

3.11. No OS kernel modification

The decision to avoid OS kernel modifications has facilitated scaling across a variety of platforms, and operating systems, particularly enabling implementation of Movers on platforms from several vendors.

4. Scaling I/O throughput using Scalable Units and system tuning

Scaling is a continuous process as new hardware and media are introduced, as storage requirements increase, and as improved device capacities and performance enter the marketplace. Successfully accomplishing a high speed transfer is an end-to-end problem that requires the interaction of a myriad of components ranging from platform operating systems and file systems through choice of network buffer, window and packet sizes, to network switching components and topology, and ultimately down to the microcode in target device controllers. Often the direction of the transfer has profound effects on transfer speeds. The transfer is, of course, only as fast as the slowest component.

Finding the sweet spot for an entire I/O chain is difficult. HPSS allows system administrators to adjust a variety of parameters to tune transfer performance. Example parameters include the storage segment size and number, virtual volume block size (used in striping), packet size, Mover buffer size, TCP flow control window size, and numbers of NICs. Sites typically determine an optimal combination consisting of a Mover with a set of devices such as disk drives, controllers, channels, Mover nodes, Mover network connections and associated HPSS and interface parameters to provide high bandwidth transfers to/from large compute platforms. This combination is labeled a Scalable Unit. An example of a disk Scalable Unit would include a particular Mover platform (model, number of CPUs, memory, NICs, HBAs) attached to a RAID array, with an appropriate Logical Unit organization, providing a tuned data throughput for a fixed disk capacity.

As data throughput requirements grow or improved storage devices become available, sites simply add repetitive or new types of Scalable Units to satisfy the requirement. As technologies advance, a site constantly tunes, identifies and migrates to new Scalable Unit types. This is made possible by the modularity of HPSS, particularly the separation of Movers and their network connections from other metadata modules. Being able to add virtually unlimited numbers of Movers, and thus the above Scalable Units, is a key scalability feature.

As examples of the importance of proper tuning, we have seen performance improvements of 10X and even more by tuning the network and HPSS buffer sizes. We have seen rates go from 12MB/s to 80+MB/s simply by changing the network send/receive socket buffer size. We have seen 2X to 4X improvement merely by properly choosing virtual volume block sizes, and Mover buffer sizes. We saw large improvements for earlier IBM AIX operating system versions when it was discovered that 32K is the sweet spot for doing writes to the network. Many more examples could be given.

5. How HPSS achieves scalability across the scaling dimensions

In this section we go through each of the scalability dimensions defined in Section 1. We identify the key architectural and implementation features outlined above and briefly recap how they are achieved. We also discuss areas requiring further evolution, both to better meet current requirements and future scalability.

5.1. Scalable data throughput

Architecture: The architectural features proven successful in I/O scaling have been the separation of data movement from Metadata Services and the use of Movers to virtualize storage devices, memory and files to create virtual network-attached storage devices or make storage devices directly accessible to clients on SANs. Thus, by expanding network cross-sectional bandwidth and replicating Scalable Units as needed, one can meet a given I/O throughput requirement. The Storage Server and its virtual volume (e.g. striped volumes) and virtual segment services have also aided scalable data throughput.

Implementation: The important implementation choices supporting I/O scalability include: the ability to support many concurrent requests and I/Os in progress, a rich set of communications and client transfer agent capabilities at several levels, allowing balanced use of networking resources (types of networks, connections, NICs, nodes, Movers), the use of device striping, extended file attributes that can be used to optimally schedule tape mounting, COSs, configuration files usable by client agents and Movers, and a scalable metadata engine.

Deployment: Review of metadata engine utilization on current equipment is a strategy that applies to this and all dimensions below to determine if workstation capacity should be upgraded, replaced, or the metadata should be restructured across multiple subsystems/workstations to improve service. The ability to use faster commodity processors and multiprocessors for scaling Metadata Service and Mover I/O performance is an important and cost effective strategy. Periodic planning is important to determine I/O requirements and associated networking, Mover and device resources needed to meet these requirements. The specification of Scalable Units (Movers and attached devices) and end-to-end tuning of the appropriate network, communication, device configuration and HPSS parameters have been an essential part of the I/O scalability strategy. Defining COSs and File Families and their mapping to the appropriate hierarchies has also been useful in organizing the storage devices for optimal I/O.

Issues needing further work: There is a need to improve small file performance (e.g. the number of small file creates and/or writes/sec). This will be a focus of near term work. We are studying several areas of the implementation where we can improve small file performance: reducing metadata; better use of DB2's rich functionality for metadata organization, access, caching, and partitioning for better concurrent access; improving Storage Service and Mover communications; and improving tape organization and small file aggregation. Many of these will improve metadata performance for all other operations as well. The Storage Service storage allocation algorithm can also be improved to more uniformly balance disk utilization across multiple Movers to take advantage of knowledge of which disks are busy and which are idle. Making Subsystems easier to use is also an option for distributing metadata services for improved performance.

5.2 Scalable storage capacity and storage space management

Architecture: The hierarchical storage architecture with appropriate choice of abstract/physical objects, and object management modularity has been central to scaling capacity.

Implementation: Important implementation choices include: metadata engine choice and scalable metadata design and organization; wide use of 64 bit fields for essentially unlimited numbers and sizes of objects; multiple storage hierarchies, COS, and File Families; separation of migration/purge policy and mechanism supporting site specific policies; a repack utility for both reclaiming tape space and moving data to newer technologies for technology insertion.

Deployment: Periodic storage requirements and capacity planning, including review of new technologies to replace current disk/tape resources leverage the ability of HPSS to easily scale to meet identified requirements.

Issues needing improvement: None currently identified.

5.3 Scalable robustness

Architecture: Separation of metadata and user data storage allows multiple redundancy mechanisms to be employed as appropriate to meet reliability, availability and recoverability requirements.

Implementation: Successful approaches include: use of a commercial quality RDBMS metadata engine, metadata backup, mirroring and recovery, atomic transactions, diagnostic utilities, the ability to create hierarchies and COS that make multiple copies of user data on tape, and use of the migration/purge policies to provide "automatic" backup. Use of backup metadata engines and, where appropriate, the IBM High Availability Option [16] that provides hot backup and recovery.

Deployment: Strategies include: careful attention to setting up redundant metadata systems and storage, well defined backup and recovery procedures, metadata engine documentation and support coverage.

Issues needing improvement: None currently identified.

5.4 Scalable name service

Architecture: Distributed architecture supporting a distributed, shared global name space.

Implementation: Strategies include: a powerful metadata system allowing named objects to be located rapidly using a variety of search criteria, large field sizes in metadata records, scalable algorithms that support operation times that are independent of directory size or for "directory list" linear time with directory size, support for Filesets and Junctions allowing linking name spaces across HPSS systems.

Deployment: No special strategies in use.

Issues needing improvement: Metadata performance improvements as listed earlier.

5.5 Scalable numbers of clients

Architecture: Separation of Metadata Service and data transfer supporting many simultaneous transfers and the ability to integrate HPSS with user systems such as file systems and web servers.

Implementation: Important implementation decisions include metadata engine choice, supporting scalable concurrency through widespread use of threads and thread safety for multiprocessing.

Deployment: An example of both use with large numbers of clients and geographical deployment is the HPSS environment at Indiana University, mentioned above.

Issues needing improvement: Ongoing metadata performance improvements and completing a DFS replacement plan as described above.

5.6 Scalable deployment across geographical distances and cooperating institutions

Architecture: Key features include the modular distributable architecture.

Implementation: The ability to distribute and join name spaces and join distributed and multiple security domains.

Deployment: Examples of distributed HPSS usage is the "single user logon" access to any of the three systems at LLNL, SNL and LANL and the Indiana University system mentioned above. Besides the distributed users, Indiana University is an example of using the HPSS capability to distribute their storage devices so as not to be limited by WAN bandwidth for local access. They use a single HPSS metadata service for the entire system to create a global name space and single virtual shared environment, but have distributed Movers and Physical Volume Repositories for distributed disks, robots and tape drives [11].

Issues needing improvement: None currently identified.

5.7 Scalable storage system management

Architecture: HPSS Storage System Management consists of the component mentioned earlier and a set of utilities. The modular architecture and the auto-generation of screens facilitate organization of the storage system management metadata, internal communications, and the user interface. These make the addition of new features faster and easier.

Implementation: Strategies include: concurrency and scalable metadata engine; support for day-to-day operations using either an easily navigable GUI or command line interface; support for operations across large numbers of servers or devices using the command line interface and associated scripts (e.g. setting up initial configurations); and a set of utilities for troubleshooting, recovery and tuning. The task of developing and maintaining GUI screens is eased through the use of screen auto-generation techniques.

Deployment: Different sites utilize the GUI and command line interfaces in different ways and combinations. For example, some sites deploy 6 – 12 concurrent GUI users, each with 5 – 20 open screens, on a variety of Storage System Management desktops (Linux, Windows, Macintosh). Tests have shown that, should the need arise, many more concurrent users and active management screens can be supported. Other sites use the command line interface in scripts to perform regular system monitoring. The command line interface is also used for managing HPSS systems remotely. Flexibility exists with HPSS to manage distributed resources and HPSS systems as independent entities while linking them via the option of the federated namespace, or by using a single HPSS system to manage remote resources (Remote Mover option) such as at Indiana University or to manage remotely placed HPSS subsystems.

Issues needing improvement: Ongoing evolution of functionality and presentation screens, including supporting all commands in the command line interface, increasing the number and usefulness of the "views" of the system available, particularly the addition of screens which tie together related information (e.g. to track a user's job as it flows through the system).

5.8 Scalable security

Architecture: Ability to integrate a distributed modular system into a secure environment using local and cross-site security infrastructures and modular organization of security components into plug and play security mechanisms.

Implementation: Choices useful here include: common security infrastructure mechanisms (e.g. Kerberos, GSS, LDAP, Unix permissions), separation of policy and mechanism, the current version of LDAP (IBM's) scales in the same manner as HPSS since it uses DB2. With Kerberos, one can create multiple security realms.

Deployment: Each site has different institutional policies and possibly different directory products or implementations and care is required to appropriately set these up.

Issues needing improvement: Improving the deployment procedures for secure cross realm integration.

5.9 Client roles in scalability

Architecture: The two features provided for building parallel-capable interfaces and powerful client agents are a single parallel API (CLAPI) and a structured configuration file. A range of capabilities can be built on these, depending on what the customer requires. The users have the option to either use a transparent interface (e.g., Linux VFS or XDMS with XFS) or use a powerful client agent (e.g. PFTP/FTP, HTAR, HSI, PSI), or a combination of these. For example, client agents have the ability to automatically make use of multiple NICs and client nodes, parallel I/O capabilities, network tuning and network configurations, select proper COS based upon file characteristics or user-specified options, optimize tape mounts for file retrievals, all with goal of transparently providing all of the power of HPSS via a simple intuitive interface that does not require the user to understand the system complexities.

Implementation: A rich and growing set of user interface capabilities: CLIAPI library, VFS, client agents (e.g. PFTP, HSI, HTAR, PSI and others), extended file attributes and configuration files that support transfer scheduling and optimization, use of front-end file systems such as was done with DFS at the Indiana University and XFS integrated with XDSM.

Deployment: HPSS provides a variety of user interfaces mentioned above. Alternatively sites can build custom interface applications. For example, the European Centre for Medium Range Weather Forecasts (ECMWF) has made extensive use of the CLAPI library to build high performance applications customized to its special requirements.

Issues needing improvement: Ongoing evolution, standardization and addition of scalable user interface functionality and production deployment of the transparent VFS interface.

6. Example storage and I/O throughput rate data

In this section we present examples of scalability histories from selected HPSS sites. Numbers quoted are as of December 2004.

6.1 I/O scaling examples

LLNL - Aggregate data transfer rates to the archive, before HPSS, were well under 10MB/s and now exceed 1.5GB/s to caching disk. Single file rates, using a four-way stripe to a RAID array, generally run at around 300 MB/s. Daily throughput to the archive has exceeded 17TB TB/day.

LANL - A recent user archive operation stored 122,000 files occupying 10TB in six hours with the transfer rate limited by network throughput. In a recent performance demonstration, a data transfer rate of 550 MB/s was achieved using 16-way mirrored tape stripes storing files over 100 GB in size on StorageTek 9940Bs.

LBNL - NERSC has gone from moving 1.5TB/day in 2001 to peak I/O days of 6TBs in 2004, with expected peak days of 10TBs in 2005. Single file transfer throughput has gone from 17MB/s in 2001 to 231MB/s in 2004, limited by network bandwidth.

BNL – Daily ingest rate from experimental devices to HPSS has reached 28TB/day, and 330MB/s and 550MB/s I/O to tape and disk respectively.

IBM - At the SC04 supercomputing conference in November 2004, IBM demonstrated HPSS (an early version of HPSS 6.2) performance using three computers, one each for HPSS, reading and writing. A large 128 GB file was written and read in 512 MB blocks using 16-way striped SAN-attached disk files, using 8 host bus adapters on each client computer. As one computer wrote each block, it was immediately read by a second computer, thus demonstrating "read behind write" performance. The file transfers were measured at 1016 MB per second on the write side and 1008 MB per second on the read side, for an aggregate data rate of just over two GB per second.

6.2 Capacity examples

LLNL - The Secure Computing Facility (SCF) held 13 TB of data in 1992 when HPSS design began. Today the SCF contains 1.5 PBs stored in about 25 million files. The Open Computing Facility (OCF), not available in 1992, today contains an additional 1.3 PB stored in about 18 million files. There are about 1 million directories in the OCF and 0.5 million in the SCF. Directories with tens of thousands of entries are in use in both SCF and OCF, the largest with about 90,000 entries.

Other sites known to be storing a petabyte or more in HPSS include:

2.3PB Los Alamos National Laboratory, in 33M files.
2+PB: Brookhaven National Laboratory (BNL).

1+PB: Commissariat à l'Energie Atomique/Division des Applications Militaires (CEA/DAM) Compute Center in France.
2+PB: The European Centre for Medium-Range Weather Forecasts (ECMWF) in England.
1+PB: National Energy Research Scientific Computing Center (NERSC), in 33M files.
1.5PB: San Diego Supercomputer Center (SDSC).
1.4PB: Stanford Linear Accelerator Center (SLAC).
Many sites, such as ORNL, are doubling their stored data yearly and will also shortly reach a petabyte.

7. Conclusion

HPSS design, implementation, and deployment have resulted in a robust, scalable system that has successfully met its initial scaling goals in the several dimensions discussed above. Particularly visible are I/O and capacity scaling. Based on experience at HPSS sites such as LLNL and through demonstrations, today's performance compared to performance at the time the HPSS project began, has demonstrated the following scaling factors:
100 for capacity to petabytes,
1000 for instantaneous throughput to GB/s,
1000 for daily throughput to 10s TB/day, and
1000 for single file bandwidth to GB/s.

The HPSS system architecture and implementation is such that there is lots of room for further scaling in these important I/O and capacity dimensions by further orders of magnitude into the future (e.g. extending its capacity to 100s of petabytes to exabytes stored and I/O throughput to 100s gigabytes/sec to terabytes/sec). The other dimensions of scalability discussed will also continue to scale. The modular network centric, distributable architecture of HPSS and modular industry standard product infrastructure are sound. The major near term focus will be on measurement, tuning and optimization, particularly metadata performance, outlined in Section 5, thus improving small file performance and supporting other scalability dimensions.

Acknowledgement

We wish to thank the many developers within the HPSS Collaboration who have created HPSS and provided helpful comments on this paper. This work was, in part, performed by the Lawrence Livermore National Laboratory, Los Alamos National Laboratory, Oak Ridge National Laboratory, National Energy Research Supercomputer Center and Sandia National Laboratories under auspices of the U.S. Department of Energy, and by IBM Global Services – Federal.

References

All URLs in references below were tested and working on Jan. 6, 2005.

HPSS references organized by date

[1] A.L. Buck and R.A. Coyne, Jr., ``Dynamic Hierarchies and Optimization in Distributed Storage System," *Digest of Papers, 11th IEEE Symp. Mass Storage Systems, Oct.* 7-10, 1991, IEEE Computer Society Press, 85-91.
[2] R. Coyne, H. Hulen, and R. Watson, ``The High Performance Storage System," *Proc. Supercomputing '93*, November 1993, 15-19.
[3] J.K. Deutsch and M.R. Gary, " Physical Volume Library Deadlock Avoidance in a Striped Media Environment," *Proc. IEEE MSS Symposium*, IEEE Computer Society Press, 1995.
[4] S. Louis and D. Teaff, ``Class of Service in the High Performance Storage System," *Proc. 3rd IFIP TC6 International Conf. Open Distributed Processing*, Brisbane, Australia, Feb. 1995, 21-24.
[5] Danny Teaff, Dick Watson, and Bob Coyne, "The Architecture of the High Performance Storage System (HPSS)," *Proceedings of the Goddard Conference on Mass Storage & Technologies*, College Park, Maryland, March 1995.
[6] R. Watson and R. Coyne, "The Parallel I/O Architecture of the High Performance Storage System (HPSS)," *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems*. IEEE Computer Society Press, September 1995, 27-44.
[7] D. Fisher and T. Tyler, ``Using Distributed OLTP Technology in the High Performance Storage System," *Proc. 14th IEEE Computer Society Mass Storage Systems Symp.*, Monterey, CA, Sept. 11-14, 1995.
[8] Per Lynse, Gary Lee, Lynn Jones, and Mark Roschke. "HPSS at Los Alamos: Experiences and analysis," *Proc. Sixteenth IEEE Symposium on Mass Storage Systems in cooperation with the Seventh NASA Goddard Conference on Mass Storage Systems and Technologies*, IEEE, March 1999, 150-157.
[9] Harry Hulen, Otis Graf, Keith Fitzgerald and Richard W. Watson, "Storage Area Networks and the High Performance Storage System," *Proceedings Goddard Mass Storage Conference*, April 2002.
[10] M. Gleicher, Hierarchical Storage Interface and HTAR Web sites, (HSI) http://www.hpss-collaboration.org/hpss/HSI/index.html, (HTAR) http://www.llnl.gov/LCdocs/htar/.
[11] Haichuan Yang, "Building a Massive Data Storage Infrastructure for the Masses", *Supercomputing 2003 Conference*, Baltimore, MD, November 14-18, 2002. http://storage.iu.edu/presentations/index.html.

Other references

[12] ADIC, StorNext File System, http://www.adic.com/ibeCCtpItmDspRte.jsp?section=10024&item=121889.
[13] Cluster File Systems, Lustre a Scalable High-Performance File System, White Paper, http://www.lustre.org/documentation.html.

[14] Dennis Colarelli, Dirk Grunwald, "Massive arrays of idle disks for storage archives," *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Baltimore, Maryland, November 16 - 22, 2002, 1–11.
[15] IBM, Distributed File System, http://www-306.ibm.com/software/stormgmt/dfs/.
[16] IBM, High availability option, HACMP for AIX 5L, http://www-1.ibm.com/servers/aix/products/ibmsw/high_avail_network/hacmp.html.
[17] IBM Total Storage: Introducing the SAN File System SAN FS, http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg247057.html?Open.
[18] IBM, DB2 Universal Database for Linux, Unix and Windows, http://www-306.ibm.com/software/data/db2/udb/.
[19] IEEE Mass Storage System Reference Model V5 (MSSRM), available at http://www.ssswg.org/public_documents.html.
[20] IEEE, IEEE Standard 1003.1-1988, Portable Operating System Interface for Computer Environments", 1988.
[21] IEEE, POSIX threads, ISO/IEC standard 9945-1:1996, IEEE publications catalogue number SH 94352-NYF.
[22] Kline, B., Distributed File Systems for Storage Area Networks, http://hsi.web.cern.ch/HSI/HNF-Europe/SEM3_2000/DistFileSystems.pdf.
[23] J. Kohl, C. Neuman, Request for Comments: RFC 1510, The Kerberos Network Authentication Service (V5), Sept. 1993.
[24] J. Linn, Request for Comments: RFC 1508, Generic Security Service Application Program Interface, September 1993.
[25] R. O'Donnell, "Holographic Data Storage Systems," *Proceedings of the IEEE*, Volume: 92, Issue: 8, Aug. 2004, 1229- 1230.
[26] Panasas, Panasas Object Storage Architecture, www.panasas.com/docs/Object_Storage_Architecture_WP.pdf.
[27] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, D. Noveck, Request for Comments: 3530, "Network File System (NFS) version 4 Protocol," April 30, 2003.
[28] M. Sherman, "Architecture of the Encina distributed transaction processing family, *"Proceedings of the 1993 ACM SIGMOD international conference on Management of data,* 1993, 460-463.
[29] SGI, XFS: A high-performance journaling file system, http://oss.sgi.com/projects/xfs/.
[30] R. Srinivasan, RPC: Remote Procedure Call Protocol Specification Version 2, Network Working Group Request for Comments: 1831, August 1995.
[31] Sun Microsystems, Sharing Data with Sun StorEdge™ Performance Suite: QFS, www.sun.com/storage/white-papers/sharing_data_qfs.pdf.
[32] Chris Wood, "Client/Server Data Serving for High-Performance Computing, "*Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems*, IEEE Computer Society Press, September 1995.
[33] X/Open, Systems Management: Data Storage Management (XDSM) API, X/Open Document Number: C429, http://www.opengroup.org/onlinepubs/9695979099/toc.htm
[34] X/Open, Distributed Computing Environment, http://www.opengroup.org/tech/dce/.
[35] W. Yeong, T. Howes, S. Kille, Request for Comments: RFC 1777 Lightweight Directory Access Protocol, March 1995.

This work was performed under the auspices of the U.S. DOE by UC, LLNL under contract No. W-7405-Eng-48.
UCRL-PROC-208872