, 175 / 1-4693

N95-24113

The Architecture of the High Performance Storage System (HPSS)

Danny Teaff
IBM Federal
3700 Bay Area Blvd.
Houston, TX 77058
(713) 282-8137
Fax (713) 282-8074
teaff@vnet.ibm.com

S5-82
43449
p- 29

Dick Watson
Lawrence Livermore National Laboratory
PO Box 808, L-560
Livermore, CA 94550
(510) 422-9216
Fax (510) 423-7997
dwatson@llnl.gov

Bob Coyne
IBM Federal
3700 Bay Area Blvd., 5th Floor
Houston, TX 77058
(713) 282-8039
Fax (713) 282-8074
coyne@vnet.ibm.com

Abstract

The rapid growth in the size of datasets has caused a serious imbalance in I/O and storage system performance and functionality relative to application requirements and the capabilities of other system components. The High Performance Storage System (HPSS) is a scalable, next-generation storage system that will meet the functionality and performance requirements of large-scale scientific and commercial computing environments.

Our goal is to improve the performance and capacity of storage systems by two orders of magnitude or more over what is available in the general or mass marketplace today. We are also providing corresponding improvements in architecture and functionality. This paper describes the architecture and functionality of HPSS.

Introduction

The rapid improvement in computational science, processing capability, main memory sizes, data collection devices, multimedia capabilities, and integration of enterprise data are producing very large datasets. These datasets range from tens to hundreds of gigabytes up to terabytes. In the near future, storage systems must manage total capacities, both distributed and at single sites, scalable into the petabyte range. We expect these large datasets and capacities to be common in high-performance and large-scale national information infrastructure scientific and commercial environments. The result of this rapid growth of data is a serious imbalance in I/O and storage system performance and functionality relative to application requirements and the capabilities of other system components.

To deal with these issues, the performance and capacity of large-scale storage systems must be improved by two orders of magnitude or more over what is available in the general or mass marketplace today, with corresponding improvements in architecture and functionality. The goal of the HPSS collaboration is to provide such improvements. HPSS is the major development project within the National Storage Laboratory (NSL). The NSL was established to investigate, demonstrate, and commercialize new mass storage system architecture to meet the needs above [5,7,21]. The NSL and closely related projects involve more than 20 participating organization from industry, Department of Energy (DOE) and other federal laboratories, universities, and National Science Foundation (NSF) supercomputer centers. The current HPSS development team consists of IBM U.S. Federal, four DOE laboratories (Lawrence Livermore, Los Alamos, Oak Ridge, and Sandia), Cornell University, and NASA Langley and Lewis Research Centers. Ampex, IBM, Maximum Strategy Inc., Network Systems Corp., PsiTech, Sony Precision Graphics, Storage Technology, and Zitel have supplied hardware in support of HPSS development and demonstration. Cray Research, Intel, IBM, and Meiko are cooperating in the development of high-performance access for supercomputers and MPP clients.

The HPSS commercialization plan includes availability and support by IBM as a high-end Service offering through IBM U.S. Federal. HPSS source code can also be licensed and marketed by any US. company.

Architectural Overview

The HPSS architecture is based on the IEEE Mass Storage Reference Model: version 5 [6,9] and is network-centered, including a high speed network for data transfer and a separate network for control (*Figure 1*) [4,7,13,16]. The control network uses the Open Software Foundation's (OSF) Distributed Computing Environment DCE Remote Procedure Call technology [17]. In actual implementation, the control and data transfer networks may be physically separate or shared. An important feature of HPSS is its support for both parallel and sequential input/output (I/O) and standard interfaces for communication between processors (parallel or otherwise) and storage devices. In typical use, clients direct a request for data to an HPSS server. The HPSS server directs the network-attached storage devices or servers to transfer data directly, sequentially or in parallel to the client node(s) through the high speed data transfer network. TCP/IP sockets and IPI-3 over High Performance Parallel Interface (HIPPI) are being utilized today; Fibre Channel Standard (FCS) with IPI-3 or SCSI, or Asynchronous Transfer Mode (ATM) will also be supported in the future [3,20,22]. Through its parallel storage support by data striping HPSS will continue to scale upward as additional storage devices and controllers are added to a site installation.

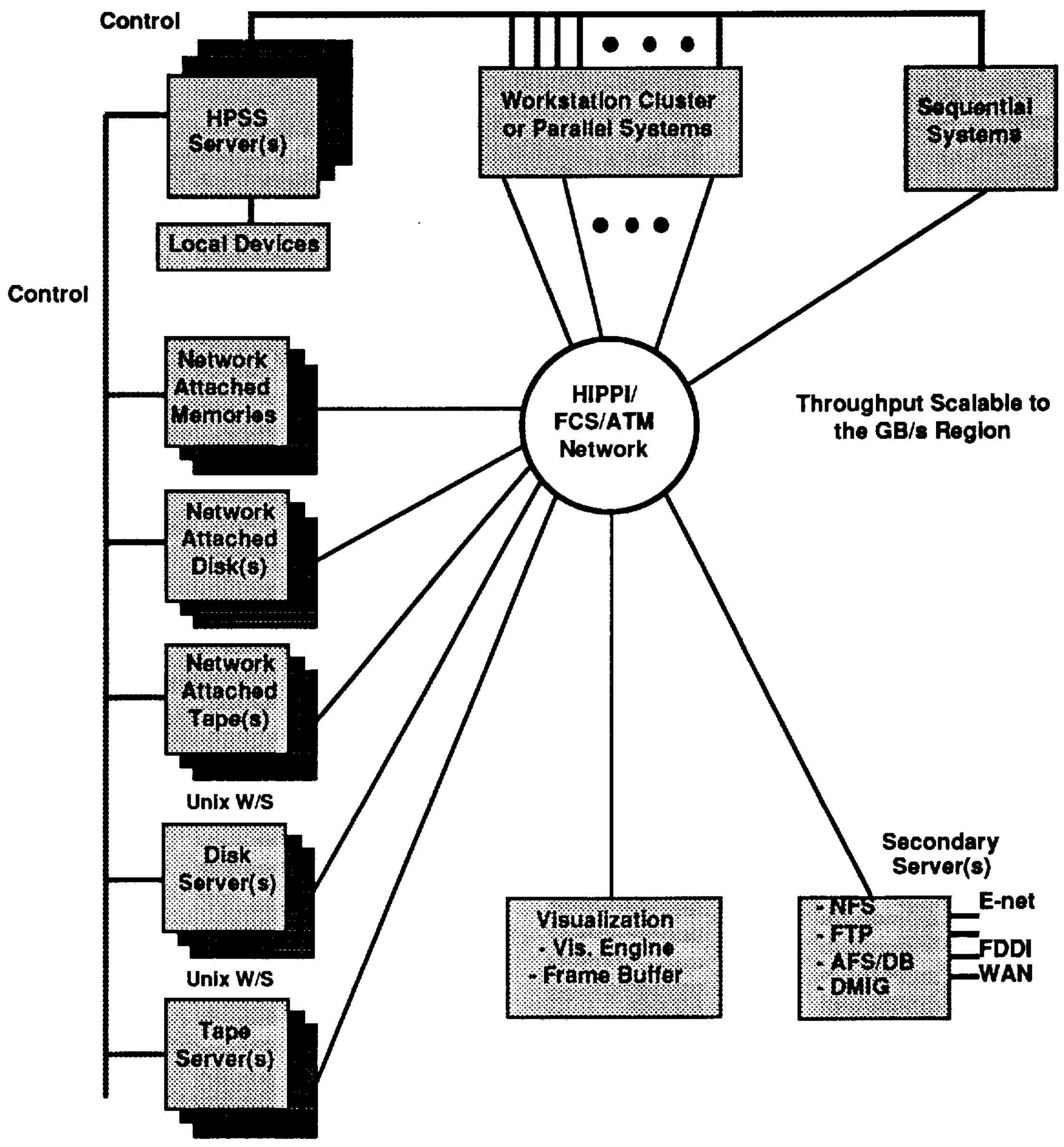


Figure 1 - Example of the type of configuration HPSS is designed to support

The key objectives of HPSS are now described.

Scalability

A major driver for HPSS is to develop a scalable, distributed, high performance storage management system. HPSS is designed to scale in several dimensions.

The HPSS I/O architecture is designed to provide I/O performance scaling by supporting parallel I/O through software striping [1]. The system will support application data transfers from megabytes to gigabytes per second with total system throughput of many gigabytes per second. Data object number and size must scale to support billions of data objects, each potentially terabytes or larger in size, for total storage capacities in petabytes. This is accomplished through 64-bit metadata fields and scalable organization of system metadata. The system also is required to scale geographically to support distributed systems with hierarchies of hierarchical storage systems. Multiple storage systems located in different areas must integrate into a single logical system accessible by personal computers, workstations, and supercomputers. These requirements are accomplished using a client/server architecture, the use of OSF's DCE as its distributed infrastructure, support for distributed file system interfaces and multiple servers. HPSS also supports a scalable storage object name service capable of managing millions of directories and the ability to support hundreds to thousands of simultaneous clients. The latter is achieved through the ability to multitask, multiprocess and replicate the HPSS servers.

Modularity and APIs

The HPSS architecture is highly modular. Each replicable software component is responsible for a set of storage objects, and acts as a service provider for those objects. The IEEE Reference Model, on which the HPSS design is based, provides the modular layered functionality (see *Figure* 2) [6,9]. The HPSS software components are loosely coupled, with open application program interfaces (APIs) defined at each component level. Most users will access HPSS at its high level interfaces–currently client API, FTP (both parallel and sequential), NFS, Parallel File System (PFS), with AFS/DFS, Unix Virtual File System (VFS), and Data Management Interface Group (DMIG) interfaces in the future) [11,15,18,19]. However, APIs are available to the underlying software components for applications, such as large scale data management, digital library or video-on-demand requiring high performance or special services. This layered architecture affords the following advantages:

- **Replacement of selected software components**–As new and better commercial software and hardware components became available, an installation can add or replace existing components. For example, an installation might add or replace Physical Volume Repositories, Movers or the HPSS Physical Volume Library with other commercially available products.

- **Support of applications direct access to lower level services**–The layered architecture is designed to accommodate efficient integration of different applications such as digital library, object store, multimedia, and data management systems. Its modularity will enable HPSS to be embedded transparently into the large distributed information management systems that will form the information services in the emerging national information infrastructure. Support for different name spaces or data organizations is enabled through introduction of new Name Servers and data management applications.

Portability and Standards

Another important design goal is portability to many vendor's platforms to enable OEM and multivendor support of HPSS. HPSS has been designed to run under Unix requiring no kernel modifications, and to use standards based protocols, interfaces, and services where applicable. HPSS is written in ANSI C, and uses POSIX functions to enhance software portability. Use of existing commercial products for many of the infrastructure services supported on multiple-vendor platforms enables portability, while also providing market proven dependability. Open Software Foundation (OSF) Distributed Computing Environment (DCE), Transarc's Encina transaction manager [8], Kinesix SAMMI and X-windows are being used by HPSS because of their support across multiple vendor platforms, in addition to the rich set of functionality provided. The HPSS component APIs have been turned over to the IEEE Storage System Standards Working Group as a basis for its standards activities.

Reliability and Recovery

Reliable and recoverable storage of data is mandatory for any storage system. HPSS supports several mechanisms to facilitate this goal. The client-server interactions between HPSS software components have been designed to be based on atomic transactions in order to maintain system state consistency [14]. Within the scope of a given request, a transaction may be established so that an abort (or commit) in one component will cause the other participating components to abort (or commit). The HPSS Metadata Manager is fully integrated with its Transaction Manager. Following an abort, the non-volatile file and name space metadata changes within the scope of the transactions will automatically be rolled back. For recovery purposes, mirroring of the storage object and name space metadata is supported. The HPSS architecture will also support data mirroring if desired in a future release.

Support is also provided to recover from failed devices and bad media. An administrator interface is provided to place a device off line. Once the device has been repaired, it may then be placed back on line. For bad media, an application interface is provided to move storage segments from a virtual volume to a new virtual volume.

The HPSS software components execute in a distributed manner. Should a processor fail, any of the HPSS software components may be moved to another platform. Component services are registered with the DCE Cell Directory Service (CDS) so that components may locate the services. Each component has also been designed to perform reconnect logic when a connection to a peer component fails. Connection context is maintained by selected components. When a connection context is established, a keep-alive activity is started to detect broken connections. A server may use the context information associated with a broken connection to perform any necessary clean up.

Security and Privacy

HPSS uses DCE and POSIX security and privacy mechanisms for authentication, access control lists, permissions and security labels. Security policy is handled by a separate policy module. Audit trails are also supported. Further, HPSS design and implementation use a rigorous software engineering methodology which support its reliability and maintainability.

Storage System Management

HPSS has a rich set of storage system management services for operators and system administrators based on managed object definitions. The application programming interface supports monitoring, reporting and controlling operations (see Appendix A).

Software Components

The HPSS software components are shown *Figure 2*. The shaded boxes are defined in the IEEE Mass Storage Reference Model: version 5 [9].

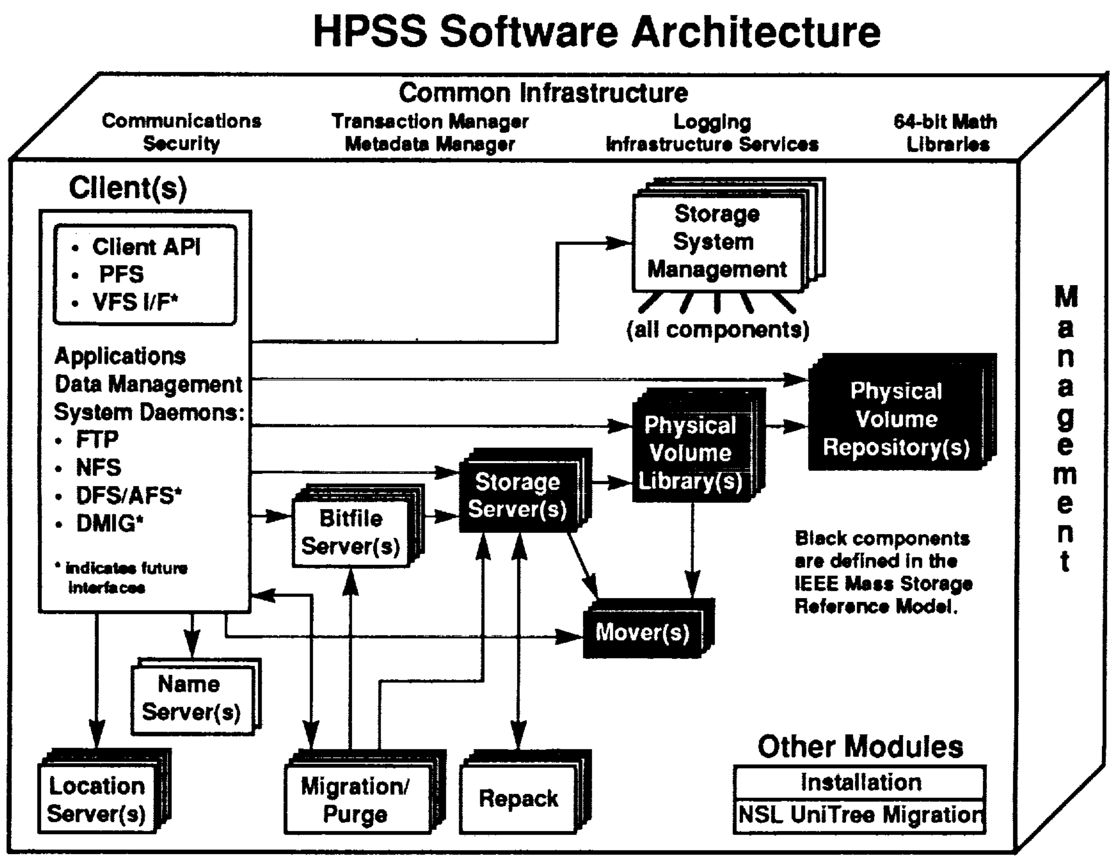


Figure 2 - Software Model Diagram

This section outlines the function of each component.

Infrastructure

HPSS design is based upon a well-formed industry standard infrastructure. The key infrastructure components are now outlined.

Distributed Computing Environment

HPSS uses OSF's DCE as the base infrastructure for its distributed architecture [17]. This standards-based framework will enable the creation of distributed storage systems for a national information infrastructure capable of handling gigabyte-terabyte-class files at gigabyte per second data transfer rates.

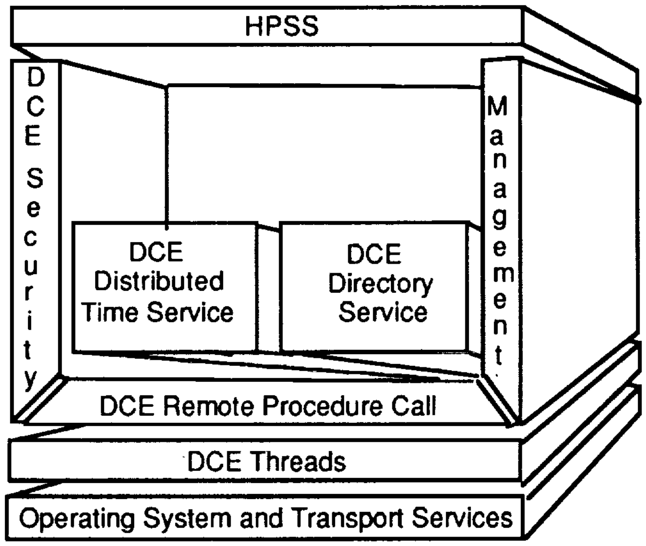

HPSS
DCE Security
Management
DCE Distributed Time Service
DCE Directory Service
DCE Remote Procedure Call
DCE Threads
Operating System and Transport Services

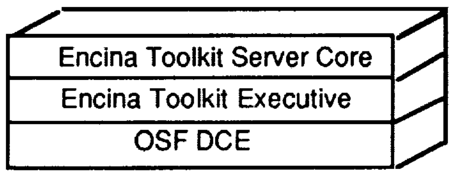
Figure 3 - HPSS DCE Architecture Infrastructure

DCE was selected because of its wide adoption among vendors and its near industry-standard status. HPSS uses the DCE Remote Procedure Call (RPC) mechanism for control messages and DCE Threads for multitasking. The DCE threads package is vital for HPSS to serve large numbers of concurrent users and to enable multiprocessing of its servers. HPSS also uses the DCE Security, Cell Directory, and Time services. A library of DCE convenience functions was developed for use in HPSS.

Transaction Management

Requests to HPSS to perform actions such as creating bitfiles or accessing file data results in client/server interactions between software components. Transaction integrity is required to guarantee consistency of server state and metadata in case a particular component should fail. As a result, a transaction manager was required by HPSS. Encina, from Transarc, was selected by the HPSS project as its transaction manager [8]. This selection was based on functionality, its use of DCE, and multi-platform vendor support.

Encina provides begin-commit-abort semantics, distributed two-phase commit, and nested transactions. In addition, Transaction RPCs (TRPCs), which extend DCE RPCs with transaction semantics, are provided. For recovery purposes, Encina uses a write-ahead log for storing transaction outcomes and updates to recoverable metadata. Mirroring of data is also provided.



```
infunction();
transaction {
  ...
}
onCommit
  ...
onAbort
  ...
```

<u>code structure</u>

Figure 4 - Encina Components

Metadata Management

Each HPSS software component has system metadata associated with the objects it manages. Each server with non-volatile metadata requires the ability to reliably store its metadata. It is also required that metadata management performance be scalable as the number of object instances grow. In addition, access to metadata by primary and secondary keys is required. The Structured File Server (SFS), an Encina optional product, was selected by the HPSS project as its metadata manager. SFS provides B-tree clustered file records, record and field level access, primary and secondary keys, and automatic byte ordering between machines. SFS is also fully integrated with the Encina transaction manager. As a result, SFS provides transaction consistency and data recovery from transaction aborts. For reliability purposes, HPSS metadata stored in SFS is mirrored. A library of metadata manager convenience functions for retrieving, adding, updating, and deleting metadata for each of the HPSS components was developed.

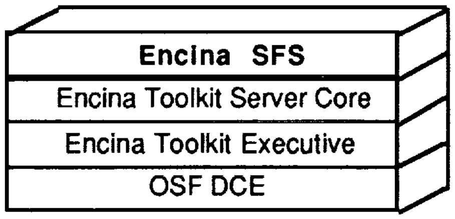


Figure 5 - Structured File Server (SFS)

Security

The security components of HPSS provide authentication, authorization, enforcement, and audit capabilities for the HPSS components. Authentication is responsible for guaranteeing that a principal is the entity that is claimed, and that information received from an entity is from that entity. Authorization is responsible for enabling an authenticated entity access to an allowed set of resources and objects. Authorization enables end user access to HPSS directories and bitfiles. Enforcement is responsible for guaranteeing that operations are restricted to the authorized set of operations. Enforcement applies to end user access to bitfiles. Audit is responsible for generating a log of security relevant activity. HPSS security libraries utilize DCE and DCE security. The authentication service, which is part of DCE, is based on Kerberos v5. The following figure depicts how HPSS security fits with DCE and Kerberos.

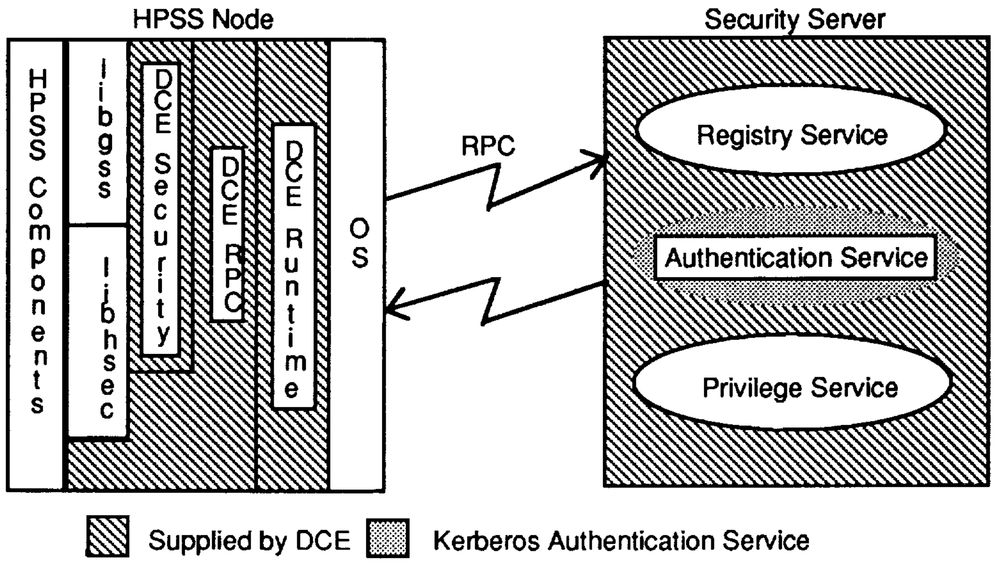


Figure 6 - HPSS Security

Communication

The control path communications between HPSS components is through DCE RPCs or Encina transaction RPCs. For data path communication, the HPSS Mover(s) currently utilize either Sockets or IPI-3 (over HIPPI) libraries. Future support is planned for IPI-3 and SCSI over Fibre Channel Standard and TCP/IP over ATM. A special parallel data transfer library has been developed. This library allows data to be transferred across many parallel data connections. The library transfers data headers that identify the data that follows. This allows data to be sent and arrive in any order on the parallel paths.

Logging

The HPSS logger is used to record alarms, events, requests, security audit records, accounting records, and trace information from the HPSS components. A central log is maintained which contains records from all HPSS components. A local log of activity from components on each HPSS node is also supported. When the central log fills, it will switch to a secondary log file. A configuration option allows the filled log to be automatically archived to HPSS. A delog function is provided to extract and format log records. Delog options support filtering by time interval, record type, server, and user.

64 Bit Arithmetic Libraries

HPSS supports file sizes up to 2**64 bytes. Many vendor platforms support only 32 bit integer arithmetic. In order to support large file sizes and large numbers of objects on 32 bit platforms, a library of 64 bit arithmetic functions has been developed. The functions support both big endian and little endian I/O architectures.

Interfaces

HPSS supports several high-level interfaces: currently Client API, FTP (both standard and parallel), and NFS, with DFS/AFS, DMIG, and VFS planned for future releases.

Client API

The HPSS Client file server API mirrors the POSIX file system interface specification where possible. The Client API also supports extensions to allow the programmer to take advantage of the specific features provided by HPSS (e.g., class-of-service, storage/access hints passed at file creation and support for parallel data transfers).

FTP (standard and parallel)

HPSS provides a standard FTP server interface to transfer files from HPSS to a local file system. Parallel FTP, an extension and superset of standard FTP, has been implemented to provide high performance data transfers to client systems. The standard FTP protocol supports third-party data transfer through separation of the data transfer and control paths, but it does not offer parallel data paths [11]. HPSS modified and augmented the standard client FTP file retrieval and storage functions to offer parallel data paths for HPSS data transfers. This approach provides high performance FTP transfers to the client while still supporting the FTP command set. Additional commands have been added to support parallel transfer. This work will be submitted to the Internet Engineering Task Force for standardization.

NFS

The NFS V2 Server interface for HPSS provides transparent access to HPSS name space objects and bitfile data for client systems from both the native HPSS and the Network File System V2 service. The NFS V2 Server translates standard NFS calls into HPSS control calls and provides data transfers for NFS read and write requests. The NFS V2 Server handles optimization of data movement requests by the caching of data and control information. If the server machine crashes, the NFS V2 Server is in charge of recovery of all cached data at the time of the crash. The NFS V2 Server will also recover when HPSS crashes. Before NFS clients can request NFS services, they must mount an exported HPSS directory by calling the Mount daemon mount API. Support for NFS V3 is planned for a future release.

Parallel File System

HPSS provides the capability to act as an external hierarchical file system to vendor Parallel File Systems (PFS). The first implementation supports the IBM SPx PIOFS. Early deployment is also planned for Intel Paragon and Meiko PFS integration with HPSS.

Name Server (NS)

The Name Server maps a file name to an HPSS object. The Name Server provides a POSIX view of the name space which is a hierarchical structure consisting of directories, files, and links. File names are human readable ASCII strings. Namable objects are any object identified by HPSS Storage Object IDs. The commonly named objects are bitfiles, directories, or links. In addition to mapping names to unique object identifiers, the Name Server provides access verification to objects. POSIX Access Control Lists (ACLs) are supported for the name space objects. A key requirement of the Name Server is to be able to scale to millions of directories and greater than a billion name space entries.

Bitfile Server (BFS)

The Bitfile Server provides the POSIX file abstraction to its clients. A logical bitfile is an uninterpreted bit string. HPSS supports bitfile sizes up to 2**64 bytes. A bitfile is identified by a Bitfile Server generated name called a bitfile-id. Mapping of a human readable name to the bitfile id is provided by a Name Server external to the Bitfile Server. Clients may reference portions of a bitfile by specifying the bitfile-id and a starting address and length. The writes and reads to a bitfile are random and the writes may leave holes where no data has been written. The Bitfile Server supports both sequential and parallel read and write of data to bitfiles. In conjunction with Storage Servers, the Bitfile Server maps logical portions of bitfiles onto physical storage devices.

Storage Server (SS)

The Storage Server provides a hierarchy of storage objects: logical storage segments, virtual volumes and physical volumes. All three layers of the Storage Server can be accessed by appropriately privileged clients. The server translates references to storage segments into references to virtual volume and finally into physical volume references. It also schedules the mounting and dismounting of removable media through the Physical Volume Library. The Storage Server in conjunction with the Mover have the main responsibility for orchestration of HPSS's parallel I/O operations.

The storage segment service is the conventional method for obtaining and accessing HPSS storage resources. The Storage Server maps an abstract storage space, the storage segment, onto a virtual volume, resolving segment addresses as required. The client is presented with a storage segment address space, with addresses from 0 to N-1, where N is the byte length of the segment. Segments can be opened, created, read, written, closed and deleted. Characteristics and information about segments can be retrieved and changed.

The virtual volume service is the method provided by the Storage Server to group physical storage volumes. The virtual volume service supports striped volumes today and mirrored volume in a future release. Thus, a virtual volume can span multiple physical volumes. The Storage Server maps the virtual volume address space onto the component physical volumes in a fashion appropriate to the grouping. The client is presented with a virtual volume that can be addressed from 0 to N-1, where N is the byte length of the virtual volume. Virtual volumes can be mounted, created, read, written, unmounted and deleted. Characteristics of the volume can be retrieved and in some cases, changed.

The physical volume service is the method provided by the storage server to access the physical storage volumes in HPSS. Physical volumes can be mounted, created, read, written, unmounted and deleted. Characteristics of the volume can be retrieved and in some cases, changed.

Repack runs as a separate process. It provides defragmentation of physical volumes. Repack utilizes a Storage Server provided function which moves storage segments to a different virtual volume.

Mover (Mvr)

The Mover is responsible for transferring data from a source device(s) to a sink device(s). A device can be a standard I/O device with geometry (e.g., a tape or disk), or a device without geometry (e.g., network, memory). The Mover also performs a set of device control operations. Movers perform the control and transfer of both sequential and parallel data transfers.

The Mover consists of several major parts: Mover parent task, Mover listen task/request processing task, Data Movement, Device control, and System Management.

The Mover parent task performs Mover initialization functions, and spawns processes to handle the Mover's DCE communication, data transfer connections, as well as the Mover's functional interface. The Mover listen task listens on a well-known TCP port for incoming connections to the Mover, spawns request processing tasks, and monitors completion of those tasks. The request processing task performs initialization and return functions common to all Mover requests. Data movement supports client requests to transfer data to or from HPSS. Device control supports querying the current device read/write position, changing the current device read/write position, loading a physical volume into a drive, unloading a physical volume from a drive, flushing data to the media, writing a tape mark, loading a message to a device's display area, reading a media label, writing a media label, and zeroing a portion of disk. System management supports querying and altering device characteristics and overall Mover state.

Physical Volume Library (PVL)

The PVL manages all HPSS physical volumes. Clients can ask the PVL to mount and dismount sets of physical volumes. Clients can also query the status and characteristics of physical volumes. The PVL maintains a mapping of physical volume to cartridge and a mapping of cartridge to PVR. The PVL also controls all allocation of drives. When the PVL accepts client requests for volume mounts, the PVL allocates resources to satisfy the request. When all resources are available, the PVL issues commands to the PVR(s) to mount cartridges in drives. The client is notified when the mount has completed.

The Physical Volume Library consists of two major parts: Volume mount service and Storage system management service.

The volume mount service is provided to clients such as a Storage Server. Multiple physical volumes belonging to a virtual volume may be specified as part of a single request. All of the volumes will be mounted before the request is satisfied. All volume mount requests from all clients are handled by the PVL. This allows the PVL to prevent multiple clients from deadlocking when trying to mount intersecting sets of volumes. The standard mount interface is asynchronous. A notification is provided to the client when the entire set of volumes has been mounted. A synchronous mount interface is also provided. The synchronous interface can only be used to mount a single volume, not sets of volumes. The synchronous interface might be used by a non-HPSS process to mount cartridges which are in a tape library, but not part of the HPSS system.

The storage system management service is provided to allow a management client control over HPSS tape repositories. Interfaces are provided to import, export, and move volumes. When volumes are imported into HPSS, the PVL is responsible for writing a label to the volume. This label can be used to confirm the identity of the volume every time it is mounted. Management interfaces are also provided to query and set the status of all hardware managed by the PVL (volumes, drives, and repositories).

Physical Volume Repository (PVR)

The PVR manages all HPSS supported robotics devices and their media such as cartridges. Clients can ask the PVR to mount and dismount cartridges. Every cartridge in HPSS must be managed by exactly one PVR. Clients can also query the status and characteristics of cartridges.

The Physical Volume Repository consists of these major parts: Generic PVR service, and support for devices such as Ampex, STK, and 3494/3495 robot services, as well as an operator mounted device service.

The generic PVR service provides a common set of APIs to the client regardless of the type of robotic device being managed. Functions to mount, dismount, inject and eject cartridges are provided. Additional functions to query and set cartridge metadata are provided. The mount function is asynchronous. The PVR calls a well-known API in the client when the mount has completed. For certain devices, like operator mounted repositories, the PVR will not know when the mount has completed. In this case it is up to the client to determine when the mount has completed. The client may poll the devices or use some other method. When the client determines a mount has completed, the client should notify the PVR using one of the PVR's APIs. All other PVR functions are synchronous. The generic PVR maintains metadata for each cartridge managed by the PVR. The generic PVR interface calls robotics vendor supplied code to manage specific robotic devices.

The operator mounted device service manages a set of cartridges that are not under the control of a robotics device. These cartridges are mounted to a set of drives by operators. The Storage System Manager is used to inform the operators when mount operations are required.

Storage System Management (SSM)

The HPSS SSM architecture is based on the ISO managed object architecture [10,12]. The Storage System Manager (SSM) monitors and controls the available resources of the HPSS storage system in ways that conform to the particular management policies of a given site. Monitoring capabilities include the ability to query the values of important management attributes of storage system resources as well as an ability to receive notifications of alarms and other significant system events. Controlling capabilities include the ability to set the values of management attributes of storage system resources and storage system policy parameters. Additionally, SSM can request that specific operations be performed on resources within the storage system, such as adding and deleting logical or physical resources. The operations performed by SSM are usually accomplished through standard HPSS server APIs.

SSM management roles cover a wide spectrum, including configuration aspects of installation, creating new volumes, initialization, operations, and termination tasks. SSM can provide management capabilities to a range of clients, including site administrators, systems administrators, operations personnel, complex graphical user interface (GUI) management environments, and independent management applications responsible for tasks such as purges, migration, and reclamation. Some of the functional areas of SSM include fault management, configuration management, security management, accounting management, and performance management.

SSM consists of these major parts: SSM Graphical User Interface (SAMMI GUI Displays), SAMMI Data Server, and System Manager.

The SSM Graphical User Interface allows operators, administrators, and users to interactively monitor and control the HPSS storage system. Kinesix's SAMMI product is used to provide the HPSS GUI services. SAMMI is built on X-windows and OSF's Motif. It provides mechanisms to simplify screen design and data management services for screen fields. Standard Motif widgets such as menus, scrollbar lists, and buttons are used. In addition SAMMI specific widgets such as dials, gauges, and bar charts are used for informational and statistical data.

The SAMMI Data Server is a client to the System Manager and a server to the SAMMI Runtime Display Manager. The SAMMI Data Server is the means by which data is acquired and fed to the SAMMI Displays.

The Storage System Manager is a client to the HPSS servers and a server to the SAMMI Data Server and other external clients wishing to perform management specific operations. It interfaces to the managed objects defined by the HPSS servers.

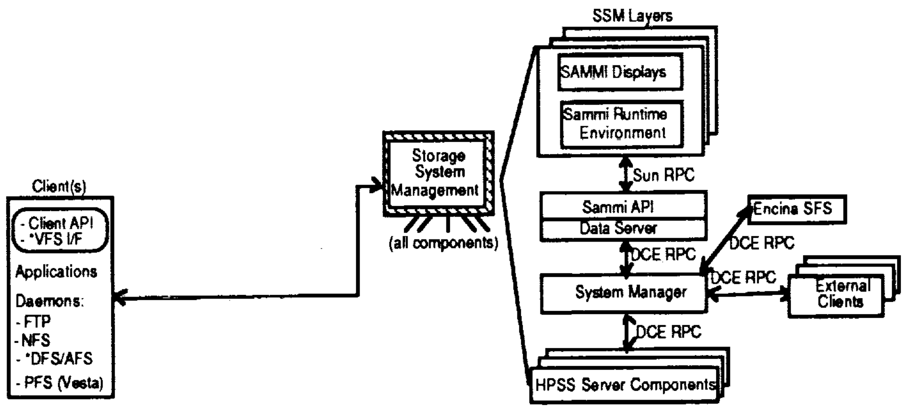


Figure 7 - Storage System Management

Migration - Purge

The Migration-Purge server provides hierarchical storage management for HPSS through migration and caching of data between devices. There are two types of migration and caching: disk migration and caching and tape migration and caching. Multiple storage hierarchies are supported by HPSS [2]. Data is cached to the highest level (fastest) device in a given hierarchy when accessed and migrated when inactive and space is required.

The main purpose of disk migration is to free up the disk storage. This type of migration contains two functions; migration and purge. Migration selects the qualified bitfiles and copies these bitfiles to the next storage level defined in the hierarchy. Purge later frees the original bitfiles from the disk storage.

The main purpose of tape migration is to free up tape volumes, and not just migrate bitfiles. The active bitfiles in the target virtual volumes are moved laterally to the free tape volumes in the same storage level. The inactive bitfiles in the target virtual volumes are migrated to the free tape volumes in the next storage level.

The HPSS component client APIs provide the vehicle for the Storage System Manger to request the server to start migration and purge whenever it is necessary. The migration-purge server is set up to run migration periodically with the time interval specified in the migration policy. In addition, the server will start the migration and purge to run automatically if the free space of a storage class is below the percentage specified in the migration-purge policy.

Other

Installation

Installation software is provided for system administrators to install/update HPSS, and perform the initial configuration of HPSS following installation. The full HPSS system is first installed to an installation node. Selected HPSS software components may then be installed (using the remote installation feature) from the installation node to the other nodes where HPSS components will be executed.

NSL-UniTree Migration

HPSS, through its support of parallel storage, provides significant improvements in I/O rates and storage capacity over existing storage systems software. In transitioning from existing systems, a migration path is required. The migration path should be transparent to end users of the storage system. The capability to migrate from NSL UniTree to HPSS is provided. The migration software handles both file metadata and actual data. Utilities convert the file metadata (e.g., storage maps, virtual volume data, physical volume data), and name space metadata from UniTree format to HPSS format. Actual data is not moved. The HPSS Mover software contains additional read logic to recognize NSL UniTree data formats when an NSL UniTree file is accessed. Utilities to support migration from other legacy storage systems will also be provided as required.

Accounting

HPSS provides interfaces to collect accounting information (initially storage space utilization). These interfaces may be used by site specific programs to charge for data storage. SSM provides user interfaces to run the accounting collection utility, change account numbers and change the account code assigned to storage objects.

Summary and Status

We have described the key objectives, features and components of the HPSS architecture. At the time this paper is being written, December 1994, HPSS Release 1 (R1) is in integration testing and planning for its early deployment at several sites has begun. R1 contains all the basic HPSS components and services and supports parallel tape. It is targeted at MPP environments with existing parallel disk services. Much of the coding for Release 2 (R2) has been completed also. R2 adds support for parallel disks, migration and caching between levels of the hierarchy and other functionality. R2 will be a complete stand-alone system and is targeted for third quarter 1995.

We demonstrated, HPSS at Supercomputing 1994 with R1 and early R2 capabilities of parallel disks, and tape access (Ampex D2, IBM NTP and 3490), to an IBM SP2, IBM RS 6000, PsiTech framebuffer, and Sony high-resolution monitor over a NSC HIPPI switch. HPSS R1 is on order 95K lines of executable source code and R2 is expected to add on another 50K lines of executable source code.

Our experience indicates that the architectural choices of basing the system on the IEEE Reference Model, use of an industry defacto standard infrastructure based on OSF DCE and Transarc Encina, and use of other industry standards such as POSIX, C, Unix, ISO managed object model for Storage System Management and standard communication protocols is sound. This foundation plus the software engineering methodology employed, we believe, positions HPSS for a long and useful life for both scientific and commercial high performance environments.

Acknowledgments

We wish to acknowledge the many discussions and shared design, implementation, and operation experiences with our colleagues in the National Storage Laboratory collaboration, the IEEE Mass Storage Systems and Technology Technical Committee, the IEEE Storage System Standards Working Group, and in the storage community. Specifically we wish to acknowledge the people on the HPSS Technical Committee and Development Teams. At the risk of leaving out a key colleague in this ever-growing collaboration, the authors wish to acknowledge Dwight Barrus, Ling-Ling Chen, Ron Christman, Danny Cook, Lynn Kluegel, Tyce McLarty, Christina Mercier, and Bart Parliman from LANL; Larry Berdahl, Jim Daveler, Dave Fisher, Mark Gary, Steve Louis, Donna Mecozzi, Jim Minton, and Norm Samuelson from LLNL; Marty Barnaby, Rena Haynes, Hilary Jones, Sue Kelly, and Bill Rahe from SNL; Randy Burris, Dan Million, Daryl Steinert, Vicky White, and John Wingenbach from ORNL; Donald Creig Humes, Juliet Pao, Travis Priest and Tim Starrin from NASA LaRC; Andy Hanushevsky, Lenny Silver, and Andrew Wyatt from Cornell; and Paul Chang, Jeff Deutsch, Kurt Everson, Rich Ruef, Tracy Tran, Terry Tyler, and Benny Wilbanks from IBM U.S. Federal and its contractors.

This work was, in part, performed by the Lawrence Livermore National Laboratory, Los Alamos National Laboratory, Oak Ridge National Laboratory, and Sandia National Laboratories, under auspices of the U.S. Department of Energy Cooperative Research and Development Agreements, by Cornell, Lewis Research Center and Langley Research Center under auspices of the National Aeronautics and Space Agency and by IBM U.S. Federal under Independent Research and Development and other internal funding.

References

1. Berdahl, L., ed., "Parallel Transport Protocol," draft proposal, available from Lawrence Livermore National Laboratory, Dec. 1994.

2. Buck, A. L., and R. A. Coyne, Jr., "Dynamic Hierarchies and Optimization in Distributed Storage System," Digest of Papers, Eleventh IEEE Symposium on Mass Storage Systems, Oct. 7-10, 1991, IEEE Computer Society Press, pp. 85-91.

3. Christensen, G. S., W. R. Franta, and W. A. Petersen, "Future Directions of High-speed Networks for Distributed Storage Environments," Digest of Papers, Eleventh IEEE Symposium on Mass Storage Systems, Oct. 7-10, 1991, IEEE Computer Society Press, pp. 145-148.

4. Collins, B., et al., "Los Alamos HPDS: High-Speed Data Transfer," Proc. Twelfth IEEE Symposium on Mass Storage Systems, Monterey, April 1993.

5. Coyne, R. A., H. Hulen, and R. W. Watson, "The High Performance Storage System," Proc. Supercomputing 93, Portland, IEEE Computer Society Press, Nov. 1993.

6. Coyne, R. A. and H. Hulen, "An Introduction to the Mass Storage System Reference Model, Version 5," Proc. Twelfth IEEE Symposium on Mass Storage Systems, Monterey, April 1993.

7. Coyne, R. A., H. Hulen, and R. W. Watson, "Storage Systems for National Information Assets," Proc. Supercomputing 92, Minneapolis, Nov. 1992, pp. 626-633.

8. Dietzen, Scott, Transarc Corporation, "Distributed Transaction Processing with Encina and the OSF/DCE", Sept. 1992, 22 pages.

9. IEEE Storage System Standards Working Group (SSSWG) (Project 1244), "Reference Model for Open Storage Systems Interconnection, Mass Storage Reference Model Version 5," Sept. 1994. Available from the IEEE SSSWG Technical Editor Richard Garrison, Martin Marietta (215) 532-6746

10. "Information Technology - Open Systems Interconnection - Structure of Management Information - Part 4: Guidelines for the Definition of Management Objects," ISO/IEC 10165-4, 1991.

11. Internet Standards. The official Internet standards are defined by RFC's (TCP protocol suite). RFC 783; TCP standard defined. RFC 959; FTP protocol standard. RFC 1068; FTP use in third-party transfers. RFC 1094; NFS standard defined. RFC 1057; RPC standard defined.

12. ISO/IEC DIS 10040 Information Processing Systems - Open Systems Interconnection - Systems Management Overview, 1991.

13. Katz, R. H., "High Performance Network and Channel-Based Storage," *Proceedings of the IEEE*, Vol. 80, No. 8, pp. 1238-1262, August 1992.

14. Lampson, B. W., M. Paul, and H. J. Siegert (eds.), "Distributed Systems - Architecture and Implementation," Berlin and New York: Springer-Verlag, 1981.

15. Morris, J. H., et al., "Andrew: A Distributed Personal Computing Environment," Comm. of the ACM, Vol. 29, No. 3, March 1986.

16. Nelson, M., et al., "The National Center for Atmospheric Research Mass Storage System," Digest of Papers, Eighth IEEE Symposium on Mass Storage Systems, May 1987, pp. 12-20.

17. Open Software Foundation, Distributed Computing Environment Version 1.0 Documentation Set. Open Software Foundation, Cambridge, Mass. 1992.

18. OSF, File Systems in a Distributed Computing Environment, *White Paper, Open Software Foundation*, Cambridge, MA, July 1991.

19. Sandberg, R., et al., "Design and Implementation of the SUN Network Filesystem," Proc. USENIX Summer Conf., June 1989, pp. 119-130.

20. Tolmie, D. E., "Local Area Gigabit Networking," Digest of Papers, Eleventh IEEE Symposium on Mass Storage Systems, Oct. 7-10, 1991, IEEE Computer Society Press, pp. 11-16.

21. Watson, R. W., R. A. Coyne, "The National Storage Laboratory: Overview and Status," Proc. Thirteenth IEEE Symposium on Mass Storage Systems, Annecy France, June 12-15, 1994, pp. 39-43.

22. Witte, L. D., "Computer Networks and Distributed Systems," IEEE Computer, Vol. 24, No. 9, Sept. 1991, pp. 67-77.

APPENDIX A

Application Programming Interfaces (APIs) to HPSS Components

HPSS provides an application client library containing file, directory, and client state operations.

The HPSS Client Library provides the following routines grouped by related functionality.

API	Clients	Description
hpss_Open	client	Optionally create and open an HPSS file
hpss_Close	client	Close a file
hpss_Umask	client	Set the file creation mask
hpss_Read	client	Read a contiguous section of an HPSS file, beginning at the current file offset into a client buffer
hpss_Write	client	Write data from a client buffer to a contiguous section of an HPSS file, beginning at the current file offset
hpss_Lseek	client	Reposition the read/write file offset
hpss_ReadList	client	Read data from an HPSS file, specifying lists for data sources and sinks
hpss_WriteList	client	Write data to an HPSS file, specifying lists for data sources and sinks
hpss_Stat	client	Get file status
hpss_Fstat	client	Get file status
hpss_Lstat	client	Get file status, returning status about a symbolic link if the named file is a symbolic link
hpss_FileGetAttributes	client	Get attributes for a file
hpss_FileSetAttributes	client	Alter file attribute values
hpss_Access	client	Check file accessibility
hpss_Chmod	client	Change the file mode of an HPSS file
hpss_Chown	client	Change owner and group of an HPSS file
hpss_Utime	client	Set access and modification times of an HPSS file
hpss_GetACL	client	Query the Access Control List of a file
hpss_DeleteACLEntry	client	Remove an entry from the Access Control List of a file
hpss_UpdateACLEntry	client	Update an entry in the Access Control List of a file
hpss_Truncate	client	Set the length of a file
hpss_Ftruncate	client	Set the length of a file

hpss_Fclear	client	Clear part of a file
hpss_Cache	client	Cache a piece of a file to a specified level in the storage hierarchy
hpss_Fcache	client	Cache a piece of a file to a specified level in the storage hierarchy
hpss_Purge	client	Purge a piece of a file from a specified level in the storage hierarchy
hpss_Fpurge	client	Purge a piece of a file from a specified level in the storage hierarchy
hpss_Migrate	client	Migrate a piece of a file from a specified level in the storage hierarchy
hpss_Fmigrate	client	Migrate a piece of a file from a specified level in the storage hierarchy
hpss_Link	client	Create a hard link to an existing HPSS file
hpss_Unlink	client	Remove an entry from an HPSS directory
hpss_Rename	client	Rename a file or directory
hpss_Symlink	client	Create a symbolic link
hpss_Readlink	client	Read the contents of a symbolic link (i.e., the data stored in the symbolic link)
hpss_Mkdir	client	Create a directory
hpss_Rmdir	client	Remove an HPSS directory
hpss_Opendir	client	Open an HPSS directory
hpss_Readdir	client	Read a directory entry
hpss_Rewinddir	client	Reset position of an open directory stream
hpss_Closedir	client	Close an open directory stream
hpss_Chdir	client	Change current working directory
hpss_Getcwd	client	Get current working directory
hpss_Chroot	client	Change the root directory for the current client
hpss_LoadThreadState	client	Updates the user credentials and file/directory creation mask for a thread's API state
hpss_ThreadCleanup	client	Cleans up a thread's Client API state
hpss_Statfs	client	Returns information about the HPSS file system
hpss_AccessHandle	client	Determines client accessibility to a file, given a Name Server object handle and file pathname
hpss_OpenBitfile	client	Opens and HPSS file, specified by bitfile ID
hpss_OpenHandle	client	Open an HPSS file, specified by Name Server object ID and, optionally, pathname

hpss_GetAttrHandle	client	Get attributes of an HPSS file, specified by Name Server object ID and, optionally, pathname
hpss_SetAttrHandle	client	Set attributes of an HPSS file, specified by Name Server object ID and, optionally, pathname
hpss_GetACLHandle	client	Query the Access Control List of a file
hpss_DeleteACLEntry-Handle	client	Remove an entry from the Access Control List of a file
hpss_UpdateACLEntry-Handle	client	Update an entry in the Access Control List of a file
hpss_LinkHandle	client	Create a hard link to an existing HPSS file, given the name space object handle of the existing object, and relative directory for the new link and the pathname of the new link
hpss_LookupHandle	client	Query the Name Server to obtain attributes, an access ticket and object handle for a specified name space entry
hpss_MkdirHandle	client	Create a new directory
hpss_RmdirHandle	client	Remove a directory
hpss_ReaddirHandle	client	Read directory entries
hpss_UnlinkHandle	client	Remove directory entry
hpss_RenameHandle	client	Rename a directory entry
hpss_SymlinkHandle	client	Create a symbolic link
hpss_ReadlinkHandle	client	Read the contents of a symbolic link
hpss_TruncateHandle	client	Set the length of a file
hpss_StageHandle	client	Stage a piece of a file to a specified level in the storage hierarchy
hpss_PurgeHandle	client	Purge a piece of a file from a specified level in the storage hierarchy
hpss_MigrateHandle	client	Migrate a piece of a file from a specified level in the storage hierarchy

The Name Server provides *APIs* for the following operations:

API	Clients	Description
ns_Insert	client	Insert a bitfile object into a directory
ns_Delete	client	Delete a name space object
ns_Rename	client	Rename a name space object
ns_MkLink	client	Create a hard link to file
ns_MkSymLink	client	Make a symbolic link
ns_ReadLink	client	Read data associated with a symbolic link
ns_GetName	client	Get path name for the specified bitfile
ns_GetACL	client	Get an ACL for the specified name server object
ns_SetACL	client	Set an ACL for the specified name server object
ns_DeleteACLEntry	client	Delete an entry from the ACL of the specified name server object
ns_UpdateACLEntry	client	Update an entry from the ACL of the specified name server object
ns_Mkdir	client	Create a directory
ns_ReadDir	client	Return a list of directory entries
ns_GetAttrs	SSM, client	Get Name Server handle and managed object attributes
ns_SetAttrs	SSM, client	Set Name Server managed object attributes

The Bitfile Server provides *APIs* for the following operations:

API	Client	Description
bfs_Create	client	Create a bitfile
bfs_Unlink	client	Unlink a bitfile
bfs_Open	client	Open a bitfile
bfs_Close	client	Close a bitfile
bfs_Read	client	Read data from a bitfile
bfs_Write	client	Write data to a bitfile
bfs_BitfileGetAttrs	SSM, client	Get bitfile managed object attributes
bfs_BitfileSetAttrs	SSM, client	Set bitfile managed object attributes
bfs_BitfileOpenGetAttrs	SSM, client	Get bitfile managed object attributes (for an open bitfile)
bfs_BitfileOpenSetAttrs	SSM, client	Set bitfile managed object attributes (for an open bitfile)
bfs_ServerGetAttrs	SSM, client	Get (common) server managed object attributes
bfs_ServerSetAttrs	SSM, client	Set (common) server managed object attributes
bfs_Copy	Migration, client	Copy storage segments for a bitfile to the next storage hierarchy level
bfs_Copy	Migration, client	Move storage segments for a bitfile to the next storage hierarchy level
bfs_Purge	Purge, client	Reclaim space (i.e., purge segments) occupied by a bitfile

The Storage Server provides *APIs* for the following operations:

API	Clients	Description
ss_BeginSession	BFS, SSM, client	Start a storage server session
ss_EndSession	BFS, SSM, client	End a storage server session
ss_SSCreate	BFS, client	Create a storage segment
ss_SSUnlink	BFS, client	Delete a storage segment
ss_SSRead	BFS, client	Read data from a storage segment
ss_SSWrite	BFS, client	Write data to a storage segment
ss_SSGetAttrs	BFS, client	Get storage segment managed object attributes
ss_SSSetAttrs	BFS, client	Set storage segment managed object attributes
ss_SSMount	Migrate, Repack, SSM, client	Mount a storage segment and assign it to a session
ss_SSUnmount	Migrate, Repack, SSM, client	Unmount a storage segment
ss_SSCopySegment	Migrate, SSM, client	Copy storage segment to new segment on different virtual volume
ss_SSMoveSegment	Migrate, Repack, SSM, client	Move storage segment to new virtual volume
ss_MapCreate	SSM, client	Create storage map for a virtual volume
ss_MapDelete	SSM, client	Delete storage map for a virtual volume
ss_MapGetAttrs	SSM, client	Get storage map managed object attributes
ss_MapSetAttrs	SSM, client	Set storage map managed object attributes
ss_VVCreate	SSM, client	Create a virtual volume
ss_VVDelete	SSM, client	Delete a virtual volume
ss_VVMount	SSM, client	Mount a virtual volume
ss_VVUnmount	SSM, client	Unmount a virtual volume
ss_VVRead	SSM, client	Read a virtual volume
ss_VVWrite	SSM, client	Write a virtual volume
ss_VVGetAttrs	SSM, client	Get virtual volume managed object attributes
ss_VVSetAttrs	SSM, client	Set virtual volume managed object attributes
ss_PVCreate	SSM, client	Create a physical volume
ss_PVDelete	SSM, client	Delete a physical volume
ss_PVMount	SSM, client	Mount a physical volume
ss_PVUnmount	SSM, client	Unmount a physical volume
ss_PVRead	SSM, client	Read a physical volume
ss_PVWrite	SSM, client	Write a physical volume
ss_PVGetAttrs	SSM, client	Get physical volume managed object attributes

ss_PVSetAttrs	SSM, client	Set physical volume managed object attributes
ss_SSrvGetAttrs	SSM, client	Get Storage Server specific managed object attributes
ss_SSrvSetAttrs	SSM, client	Set Storage Server specific managed object attributes
ss_ServerGetAttrs	SSM, client	Get (common) server managed object attributes
ss_ServerSetAttrs	SSM, client	Set (common) server managed object attributes

The Mover provides *APIs* for the following operations:

API	**Clients**	**Description**
mvr_Read	SS, PVL, client	Read data from a device or devices
mvr_Write	SS, PVL, client	Write data to a device or devices
mvr_DeviceSpec	SS, client	Load a physical volume
		Unload a physical volume
		Load message to device's display area
		Flush data to media
		Write tape mark
		Read media label
		Write media label
		Clear portion of disk
mvr_DeviceGetAttrs	SS, SSM, client	Get Mover device managed object attributes
mvr_DeviceSetAttrs	SS, SSM, client	Set Mover device managed object attributes
mvr_MvrGetAttrs	SSM, client	Get Mover specific managed object attributes
mvr_MvrSetAttrs	SSM, client	Set Mover specific managed object attributes
mvr_ServerGetAttrs	SSM, client	Get (common) server managed object attributes
mvr_ServerSetAttrs	SSM, client	Set (common) server managed object attributes

The Physical Volume Library provides *APIs* for the following operations:

API	**Clients**	**Description**
pvl_Mount	client	Synchronously mount a single volume
pvl_MountNew	SS, client	Begin creating a set of volumes to automatically mount
pvl_MountAdd	SS, client	Add a volume to the set of volumes to be mounted
pvl_MountCommit	SS, client	Mount a set of volumes
pvl_MountCompleted	PVR	Notify the PVL a pending mount has completed
pvl_CancelAllJobs	SS, SSM, client	Cancel all jobs associated with a connection handle
pvl_DismountJobId	SS, SSM, client	Dismount all volumes associated with a specific job
pvl_DismountVolume	SS, SSM, client	Dismounts a single volume
pvl_DismountDrive	SSM, client	Forces the dismount of a specified drive
pvl_Import	SSM, client	Imports a new cartridge into HPSS
pvl_Export	SSM, client	Exports a cartridge from HPSS
pvl_Move	SSM, client	Move a cartridge from one PVR to another
pvl_NotifyCartridge	PVR	Notify the PVL that a cartridge has been check in or out of a PVR
pvl_WriteVolumeLabel	SS, SSM, client	Rewrite the internal label of a specified volume
pvl_AllocateVol	SS, SSM, client	Allocate a volume to a particular client
pvl_ScratchVol	SS, SSM, client	Return a volume to the scratch pool
pvl_DriveGetAttrs	SSM, client	Get drive managed object attributes
pvl_DriveSetAttrs	SSM, client	Set drive managed object attributes
pvl_VolumeGetAttrs	SSM, client	Get volume managed object attributes
pvl_VolumeSetAttrs	SSM, client	Set volume managed object attributes
pvl_QueueGetAttrs	SSM, client	Get PVL request queue managed object attributes
pvl_QueueSetAttrs	SSM, client	Set PVL request queue managed object attributes
pvl_RequestGetAttrs	SSM, client	Get PVL request queue entry managed object attributes
pvl_RequestSetAttrs	SSM, client	Set PVL request queue entry managed object attributes
pvl_PVLGetAttrs	SSM, client	Get PVL specific managed object attributes
pvl_PVLSetAttrs	SSM, client	Set PVL specific managed object attributes
pvl_ServerGetAttrs	SSM, client	Get (common) server managed object attributes
pvl_ServerSetAttrs	SSM, client	Set (common) server managed object attributes

The Physical Volume Repository provides *APIs* for the following operations:

API	Clients	Description
pvr_Mount	PVL, client	Asynchronously mount a single volume
pvr_MountComplete	PVL, client	Notify PVL a requested mount has completed
pvr_DismountCart	PVL, client	Dismount a single cartridge
pvr_DismountDrive	PVL, client	Dismount the cartridge in a given drive
pvr_Inject	PVL, SSM, client	Accept a new cartridge into the PVR
pvr_Eject	PVL, SSM, client	Eject a cartridge from the PVR
pvr_Audit	SSM, client	Audit all or part of a repository checking external cartridge labels when possible
pvr_LocateCartridge	PVL, client	Verify whether or not a PVR manages a cartridge
pvr_SetDrive	PVL, client	Takes drives in the PVR on-line or off-line
pvr_CartridgeGetAttrs	SSM, client	Get a cartridge managed object attributes
pvr_CartridgeSetAttrs	SSM, client	Set a cartridge managed object attributes
pvr_PVRGetAttrs	SSM, client	Get PVR specific managed object attributes
pvr_PVRSetAttrs	SSM, client	Set PVR specific managed object attributes
pvr_ServerGetAttrs	SSM, client	Get (common) server managed object attributes
pvr_ServerSetAttrs	SSM, client	Set (common) server managed object attributes
pvr_ListPendingMounts	SSM, client	List all currently pending mounts for the PVR

The Storage System Manager provides *APIs* for the following operations:

API	Clients	Description
ssm_Adm	client	Perform administrative request on one or more servers (shut down, halt, mark down, reinitialize, start)
ssm_AttrGet	client	Get managed object attributes
ssm_AttrReg	client	Register an SSM client to receive notifications of data change in managed objects
ssm_AttrSet	client	Set managed object attributes
ssm_Checkin	client	Accept checkins from data server clients
ssm_Checkout	client	Accept checkouts from data server clients
ssm_ConfigAdd	client	Add a new entry to a configuration files
ssm_ConfigDelete	client	Delete an entry from a configuration file
ssm_ConfigUpdate	client	Update a configuration file entry
ssm_Delog	client	Allow accept to the delog command

ssm_DriveDismount	client	Dismount a drive
ssm_JobCancel	client	Cancel a Physical Volume Library job
ssm_CartImport	client	Import cartridges into the Physical Volume Library
ssm_CartExport	client	Export cartridges from the Physical Volume Library
ssm_ResourceCreate	client	Create resources (physical volume, virtual volume, and storage map) in the Storage Server
ssm_ResourceDelete	client	Delete resources (physical volume, virtual volume, and storage map) from the Storage Server
ssm_AlarmNotify	Logging	Receive notifications of alarms
ssm_EventNotify	Logging	Receive notifications of events
ssm_MountNotify	PVL	Receive notifications of tape mounts and dismounts
ssm_BitfileNotify	BFS	Receive bitfile data change notifications
ssm_CartNotify	PVR	Receive cartridge data change notifications
ssm_DeviceNotify	PVL	Receive device data change notifications
ssm_DriveNotify	PVL	Receive drive data change notifications
ssm_LogfileNotify	Logging	Receive log file data change notifications
ssm_MVRNotify	Mvr	Receive Mover specific data change notifications
ssm_MapNotify	SS	Receive storage map data change notifications
ssm_NSNotify	NS	Receive Name Server specific data change notifications
ssm_PVNotify	SS	Receive physical volume data change notifications
ssm_PVRNotify	PVR	Receive PVR specific data change notifications
ssm_QueueNotify	PVL	Receive PVL queue data change notifications
ssm_RequestNotify	PVL	Receive PVL request entry data change notifications
ssm_SFSNotify	Metadata Manager	Receive SFS data change notifications
ssm_SSNotify	SS	Receive storage segment data change notifications
ssm_ServerNotify	NS, BFS, SS, Mvr, PVL, PVR, Logging	Receive common server data change notifications
ssm_SsrvNotify	SS	Receive Storage Server specific data change notifications
ssm_VVNotify	SS	Receive virtual volume data change notifications
ssm_VolNotify	PVL	Receive volume data change notifications
ssm_Migrate	client	Move storage segments for a bitfile to the next storage hierarchy level
ssm_Purge	client	Reclaim space occupied by bitfiles

ssm_Repack	client	Perform defragmentation of physical volumes
ssm_MoveCart	client	Move a cartridge from one PVR to another
client_notify	client	Notify clients of alarms, events, mount requests, managed object data changes, and special System Manager requests

The following managed objects have attributes which may be queried (and set) by SSM:

Name Server	Volume
Bitfiles	Physical Volume Library queue
Bitfile Server (common)	Physical Volume Library request entry
Storage segments	Physical Volume Library server specific
Storage maps	Physical Volume Library Server (common)
Virtual volumes	Cartridge
Physical volumes	Physical Volume Repository server specific
Storage Server specific	Physical Volume Repository Server (common)
Storage Server (common)	Security server
Mover device	Log Daemon server (common)
Mover server specific	Log Client server (common)
Mover server (common)	Structured File Server
Drive	

The Storage System Manager also receives the following type of notifications from the HPSS server components:

Alarms	Tape mounts
Events	Data changes for registered object attributes

Some of the more important management operations which may be performed by the Storage System Manager include:

Import/create resources	Repack
Import cartridges	Delog
Export cartridges	Set devices online/offline
Move cartridges	Dismount drive
(from one PVR to another)	
Audit PVR	Start/stop/reinitialize/halt servers
Migrate	Configure servers
Purge	Define/modify ACLs

Migration/Purge provides *APIs* for the following operations:

API	Clients	Description
migr_StartMigration	SSM, client	Start migration for a particular storage class
migr_StartPurge	SSM, client	Start purge for a particular storage class
migr_MPSGetAttrs	SSM, client	Get the migration-purge server attributes
migr_MPSSetAttrs	SSM, client	Set the migration-purge server attributes
migr_ServerGetAttrs	SSM, client	Get (common) server managed object attributes
migr_ServerSetAttrs	SSM, client	Set (common) server managed object attributes