Now that the Storage Train's Here, Can We Get Everybody on Board?

Bob Perry
Sverdrup Technology Incorporated
Cleveland, Ohio

Abstract

The MSS product based on the IEEE Mass Storage System Reference Model has arrived, and we want it to be the focus of our MSS Services. Model specifications that might make this difficult are: too loose an affiliation of name server and bitfile server, little help in tracking operational problems, no special requirements for large files and the lack of an open programmable user interface.

This paper presents some suggestions which would make the Model more responsive to large sites and enable mass storage services to be provided by a network of MSS servers. Approaches that would simplify maintenance and growth are proposed with the understanding that "MSS" entails more qualities than just big. The Model's design should encourage the development of mass storage systems which permit sites to manage this data resource intelligently and continuously into the future.

With this goal in mind, an MSS Programmable Interface is proposed, along with how sites could use it to add their own control and adjust to standards that change in ways that may not suit their environment. Our plans to unite central and distributed servers, with the Model based MSS Server as the focus of our complex, are also described.

Many of these ideas are based on observations of our own Central File Archival and Migration Service (CFAM), which has served our Cray users since January 1989 and workstations since June 1991. This experience has taught us that the real world does not forgive omissions or flaws related to reliability, managing growth, maintenance, performance and user help, and that addressing these short comings is far easier than dealing with the resultant problems.

Fundamental MSS Applications and Staffing Concerns

The IEEE Mass Storage System Reference Model, Version 4, in the future simply referred to as "the Model", is extensive, addressing a range of topics from active and passive objects to world-wide access to bitfiles. While it states that sufficient specifications are not present to make it a standard, another missing quality is a focus on the minimal MSS requirements for a typical large site's applications. Some of these applications might include:

- Repository for shared programs, local databases and read only libraries.
- Storage support for other systems, providing backup, redundant copy and archival functions.
- Augmentation of local storage to handle very large files and/or support network file access for common file systems. This may include support for diskless workstations or specialized systems requiring high performance or additional security.
- Large data collections, imported from other sources, available to researchers in a common network format (e.g., telemetry data from space experiments).
- Storage support for part (or all) of the corporate, national or international databases for which your site is responsible.

For all of the above, with the exception of highly-volatile databases, it is beneficial for the MSS support staff to readily be able to find out:

- The source of the data: System, Owner, Import Process (if any).
- The time of original file creation, last update time, last referenced time.
- The original client's file name associated with the bitfile. (In most cases this will not change or may only change by version number.)
- The size of the bitfile, the current location of bitfile and if any copies exist.

Except for temporary data collections, MSS support groups have long term responsibility for data integrity and availability. Since most computing centers are understaffed, the MSS staff will usually need all the help it can get from clients, MSS software, hardware, and the communications support group when trouble-shooting problems, recovering data and checking out new releases. Meanwhile, the duties of maintaining storage capacity reserves and forecasting future needs for the site's MSS Complex are always present.

These applications and staffing needs lead to the first Model suggestion for consideration:

S1.0 Where possible, internal control information should also be useful for human inspection. Its format should be determined by the Model but its contents only suggested, leaving the final content decision to each MSS Complex (MC). If internal control data, such as an MC bitfile id, is made available outside the complex, it would be mapped using the Complex's address to generate a world-wide bitfile id. The format and content of world-wide control data would be specified by the Model.

This implies an address assignment similar to that for the Internet Protocol. Besides making both trouble shooting and customization for each MSS Complex much easier, this also provides a natural place to enforce different restrictions for external access.

The example in Table 1, from our CFAM Server, illustrates a "humanly inspectable" identifier. (This is not equivalent to user friendly; the end user would not normally see this or be able to reference the bitfile directly with it.) All Lewis Research Center user ids are unique and consistent across all systems; data stored on CFAM must belong to one of these user ids. (User ids are 4 to 8 characters long and unique UID numbers are associated for Unix and NFS identification.)

Table 1. Internal CFAM bitfile ID format

ccccccc.Userid.org.D$yyddd.Thhmmssx

where:

cccccccc =	CFAM ownership ID on the CFAM host storing the bitfile.
Userid =	Owner of the data collection.

org =	A code indicating the basic origin of the data:
	U$ = A Station Connected Cray System
	UT = A TCP-IP network CFAM client
	ST = Shuttle Data Imported from tape

D$ =	Arbitrary prefix.
yyddd =	Year and day that file was created.
T =	Arbitrary prefix.
hhmmss =	Time of day that the file was created.
x =	Added char. to permit unique times if multiple files are created for same user/source within a second.

This bitfile id, not our first design, is loaded with valuable information in case there is any problem with the data. Of course, all of this information and more could be determined from the name server (we call it the CFAM catalog service) but that is not as accessible. Any internally called routine processing the bitfile always has the bitfile id and can readily include it in activity logs or error messages. (This is a massive head-start when investigating problems—something we do not currently have on our UniTree system.)

The bitfile id could be encrypted if a host system needed additional security. Also compression and/or binary values in the id might be necessary to shorten it, although for us, about 40 characters seems like a bargain for the maintenance benefits a meaningful text id provides.

How Separate Should Bitfiles Be from Their Descriptors?

While the Model's reverence for separate Bitfile Servers and Name Servers is understandable on general "object oriented" principles and for how it might make connecting elemental components easier, an MSS Complex could benefit from the Bitfile Descriptor (BFDES) always being available whenever the Bitfile Data (BFDAT) is being processed. The final authority for the Bitfile Descriptor would still be the entry maintained by the master Name Server but redundant entries could be synchronized against the master with two time fields, LAST-UPD and LAST-SYNC.

A few advantages of binding the BFDES to the BFDAT:

- Bitfile Servers which are performing maintenance or migrating BFDATs to other levels might operate much faster when using their own copies of BFDESs rather than continually talking to a Name and Authorization Server. The Authorization and Synchronization may be done as an initial batch process which precedes the longer BFDAT maintenance migrations to lower hierarchical levels.

- The Name Server / Security Server / Bitfile Server and maybe even some Network Communication Interfaces should have a section to write active status information into each BFDES. Unless prevented by encryption, they should also be able to read the rest of the BFDES. Complete dependence on the Name Server might complicate the implementation of sophisticated control logic, such as, detecting and immediately flagging file corruption, choosing the best network routes, early termination of requests that encounter errors or will subsequently fail due to resource problems, and tracking access patterns.

- The BFDES must have a single-canonical representation when sent between servers. Different servers may have their own internal BFDES and only use the IEEE Model format when communicating to other servers. A "Get and Sync" function for a single BFDES, a family of BFDESs, or all BFDESs for BFDATs on a physical volume would be very useful.

- Another more specialized version of the "Get and Sync" function could be designed to import entire file systems.

Sites with very reliable and continuously available server platforms, along with equally reliable hi-speed network communications, may question some of these suggestions—with good reason. But, if you want your MC to incorporate systems that are not up one hundred percent of the time, the binding of Bitfile Descriptors to the Bitfile Data would still be of value.

To restate this as a Model suggestion:

S2.0 An MSS Complex should optionally be able to run with redundant and complete Bitfile Descriptors which always accompany the Bitfile to any Bitfile processor. Functions should be defined to synchronize redundant Bitfile Descriptors with the copy on the master Name Server and permit incorporation of data from processes which convert, transmit, store and authorize access for the BFDAT.

And a corollary to this suggestion:

S2.1 If files are archived to media outside the MSS Complex, and removed from the MC, a Bitfile Descriptor should, by default, accompany each Bitfile. (Note: this does not necessarily apply to vaulting or other media which is part of the MC storage hierarchy.) The format of the Bitfile Descriptor, along with the specification of any decryption and unpacking requirements should be defined by the Model. This specification has more world-wide and unpredictable-future-reference implications, so operational BFDES entries may have to be converted and/or augmented.

Making a Big Deal About Very Large Files!

From a Standards—with a capital "S"—viewpoint I am reluctant to include this discussion. But, as the person who most often has to resolve CFAM space shortage problems, I couldn't pass up sharing some personal observations:

- Contrary to many MSS theorists, no site has infinite disk space—although our site does have one of those theorists!
- One of the most neglected hardware areas is massive, reliable, inexpensive (i.e., cheap) direct access devices. 100 gigabytes with typical access time of .1 second and a transfer rate of 100 Kbytes per second would work. Files that could easily reside on that kind of space for years now compete with bigger and more active files. Bitfile Descriptors themselves are one of the most important classes of data which could reside there—especially those for inactive files.
- Is "big file" really a relative term? I contend that any site that has all MSS data passing through their disk cache has a similar working definition: "A file big enough to wipe out my disk cache either alone or with a couple of its friends."
- We receive many big files which will not require direct access and which, in all likelihood, may never be referenced—other than to be deleted. Such files may represent the majority of CFAM's total stored bytes (e.g., workstation backups that will only be read if a workstation experiences a file system problem; days or weeks of jet engine modeling checkpoints that are only needed if the simulation produces unexpected results; huge system and network logs kept for historical reasons and to analyze problems which may be subsequently discovered).
- Unless indirect paths yield better performance, due to network and system inequities, or produce desired redundancy, bitfile transfers should always proceed directly to/from the bitfile's preferred storage hierarchy level. (Note: "better performance" is not based exclusively on speed; reliability is of equal importance.)

While these observations have been stated somewhat humorously (I hope), the following Model suggestion may determine if an MSS Server can become the storage focus of a complex or if its use *will* be limited to specialized or very controlled applications.

S3.0 An MSS Server should be able to move bitfiles directly between clients and the storage hierarchy best suited to those files; as already stated in the Model, this could be initiated by one of the parties involved or a third party. If bitfiles must be cached, the ability of the cached space to appear as a window with simultaneous input/output must be part of the standard. Very large files, not requiring direct access, should not need to be stored completely on the disk cache.

Defining an Interface Between Mass Storage Systems

The Model describes the components of a Mass Storage System, but as hardware technology has advanced while many software directions have remained unsettled, it now seems more critical to define the interface which could connect Mass Storage Systems into a Mass Storage System Complex. The analogy to networks seems too easy, but risking mutual confusion, I'll state it anyway: there should be more concern about MSS Gateways at this time and less about MSS LAN considerations.

One of the reasons is that much bigger disk storage can temporarily overwhelm the lack of data management on a system. Typical disk storage for user workstations are now measured in gigabytes, parallel-system arrays in tens of gigabytes, and Crays in hundreds of gigabytes. In a way, such systems have become de facto, low level MSS systems, but they need a back end.

The failure of a single network file system to emerge has also made connecting that level more difficult. NFS and DFS, PC and Mac LAN Servers need back end support. Finally, mainframe systems like MVS, VM, VMS are not disappearing tomorrow or next year. When and if they are history, their databases and archival storage will in large part have to be converted and/or preserved.

The best service that an MSS standard might provide is to describe how to link disparate MSS Servers into a Complex with controlled access to the outside world. Low level MSS concerns, like how to separate Bitfiles from their Descriptors, or how much to look like a Unix directory structure is definitely missing the big picture. These requirements are certainly real, but they require less standardization and coordination.

If individual MSS Servers are reliable and inexpensive, the open architecture wisdom almost guarantees you will have them, with or without the MSS support group's blessing. Perhaps the only internal server standards that have clear external significance relate to physical volumes and devices which are less integrated into the server or may be shared, such as an STK Automated Cartridge System.

Who is the Responsible Party?

At first glance, MSS services look like file transfer and data look up, so FTP and NFS are great application interfaces? Yes and no. Yes, when everything works and valid users are making reasonable requests. No, when any of those conditions are reversed.

More study, or limited production in such an environment, reveals problems and shortcomings beyond dealing with various implementations of FTP and NFS client codes. Additional tracking and filtering of requests, backup scheduling, batch support, and unique site security requirements all require additional software and/or setup. Also, will your MSS staff accept responsibility for:

- Data integrity and accessibility for the data's lifetime,
- Some level of security and a record of attempted violations,
- Reliable transmissions,
- The identification of trouble or degraded performance, without necessarily being told of it by your users or the network support group,
- Continual service improvements with minimal user impact, including adding and/or substituting MSS Servers or server devices and interfacing new networked file system methods, and
- Controlling access to your MC from the outside world?

If they do, they will find their job is not made easier by the client (both human and machine) being responsible for data delivery and access control.

Without considering specific problems which make these access methods awkward or poor performers, the main point is that the responsibility is in the wrong place for a viable MSS service. From a standpoint of daily monitoring, checking a couple of servers is much easier than querying countless clients. Also, when control or monitoring needs to be modified, the servers would be the preferred places to make the changes.

If the Model defined a programmable interface, which included a few high-level package calls, such as for file get and put, a site could easily provide their own client interfaces. This would enable the addition of site specific security, logging, restricted access, scheduling features and site designed user interfaces. The availability of C compilers, Remote Procedure Calls, and TCP/IP on just about any potential client makes this very easy.

A site that accepts this additional programming responsibility reaps many rewards, especially if there are numerous clients. One of the principal benefits is the ability of the program interface to deal with MSS environmental changes. For example, enabling the development of real (i.e., practical) evolution paths away from individual MSS solutions on existing "bigframes" such as MVS, VM, Cray and VMS. The program interface, along with user defined sections within the Bitfile Descriptor, would permit: automatic data conversions, original or converted file names, additional authentication calls, and support for Bitfiles that are only valid on certain clients and servers.

If You're the Party, You're Suppose to Know What's Going On

If the higher levels of an MSS Complex are responsible for data integrity and delivery performance, stateful communications are required. Much of the successful history of stateless protocols has been in support of limited LAN connected systems, or for communication of non-critical information, or for environments where users were so grateful for the additional file space that they would accept flaky service.

Lower level MSS servers, for support of specialized functions and devices, and with their own embedded recovery, may make good use of stateless communications with their clients, but all of the transmissions to higher levels should use a reliable protocol. Stateful communications are part of that process, but persistence, attempted recovery after client/server failure, notification of delivery failure, and source file protection for failed transfers are also required.

Most of the concern of this paper addresses functions at the higher levels, that is, how all the MSS services can be united. Lower levels can use standard open networking to manipulate fairly current data for a controlled user population. Network paths are shorter and system administration for the low level MSS Servers can involve someone who knows something about the typical applications in use and who can help maintain reserve capacity and trouble shoot local problems.

As data migrates upward in a site's MSS Complex, it becomes more "remote" in many ways; network paths may become longer, data contents are probably more anonymous to the central support staff, files are frequently older, and how the file was stored and its anticipated access may be unknown.

A programmable interface for MSS functions would be most valuable here. Sites could require that communications with higher MSS levels must use this interface. The additional control and customization would make the process manageable by simplifying maintenance procedures, ensuring reliable communications, acquiring additional information about the data, automatically activating conversions, compression, encryption/decryption, etc.

This Model suggestion focuses on defining a programmable interface. It also describes a bound file transfer scheme to provide this higher level linkage. While I am convinced a programmable machine independent interface is needed, I don't, for a minute, think the file transfer scheme is the only solution. It is the one that we implemented so we know it works locally. Other advice, from those who have implemented world-wide databases through heterogeneous platforms, should also be sought in defining this interface.

S4.0 Define a MSS Programmable Interface (MPI) which includes calls for complete MSS file functions with reliable operation guaranteed by the client-server dialogue and the underlying network protocols used. All transfers of control information between the client and server would be in a single-canonical format, as would the data, unless the client-server negotiate another format for the data transfer.

S4.1 The MPI Complete File Functions would include:

- *CREATE-PUT, PUT-REPLACE, PUT-PARTIAL*
- *GET, GET-PARTIAL, GET-ERASE*
- *GET FILE DESCRIPTION and STATUS*
- *UPDATE FILE DESCRIPTION and STATUS*
- *RENAME FILE and UPDATE FILE DESCRIPTION*
- *LOCK and SET TIME-of-LOCK, RELOCK and SET TIME-of-LOCK*
- *UNLOCK and SET TIME-of-UNLOCK*
- *SYNC FILES and SET TIME-of-SYNC*
- *ACL CONTROL*

S4.2 The Bitfile Descriptor and Control Data Structure that is provided with each Complete File Function would include all calling parameters required by the function. This structure would also be returned with some of the fields filled in by the server. Some of these parameters would be:

- Function Requested and Function Modifiers
- Return Codes from Server
- Owner ID
- New owner ID
- Complete File Name (origin system type dependent)
- Complete New File Name (new system type dependent)
- Caller Bitfile ID (if any) Server Bitfile ID
- Identifier of Master MSS Level for Bitfile. If not specified explicitly the Server handling the call will become the Master MSS Level.
- Some of the standard attributes and control information you would expect, and some you might not:
    - original size, original size known-or-estimated indicator, compressed size, file type, family association
    - source of data (may not be the same as the client storing it)
    - conversions required
    - compression/encryption required
    - MSS Access Control List or Coded Identifier of ACL

- times: created, last-updated, last-referenced, locked, un-locked, last synchronized with Master MSS level, expiration
- Current activity control: counts and last opened by identifier for both read-only, create-update and MSS maintenance callers
- Historical activity: counts for read-only, update and MSS maintenance calls; last known number of duplicates;
- Security and Authentication requirements
- Storage hierarchy elements containing the Bitfile on the client system (e.g., disks, hi-performance disk, cartridge type, network)
- Storage hierarchy elements containing the Bitfile on the server
- Acceptable server storage hierarchy levels
- User supplied text description associated with data
- Additional text and meta data associated with permanently archived files

These suggestions are just a starting point. Many of the fields would vary in format based on origin systems, security requirements and performance requirements. An organization should want to make these site policy decisions and incorporate them programmatically. For example, if a filename is going to be altered or an alias assigned, you would certainly want it done consistently and not vary from one FTP client to another, or if data is being archived permanently, more information about it contents, source and reference point may be required.

All Aboard! Current and Potential Clients

Our expanding use of CFAM and the protracted development of UniTree, along with the continual improvements in storage hardware have made us question terms such as "interim", "ultimate" and "permanent". Instead we favor a process that enables the ongoing incorporation of industry standard resources into an evolving MSS Complex. CFAM and our Distributed Storage Servers are production elements in our MC while UniTree is currently a limited-production element. DFS and Kerberos are potential new elements. Our goal is not just to improve and unify storage services but also to integrate new elements with minimal impact on existing production services.

Our centrally supported MSS Complex, outlined in the next section, is still in its infancy, while its potential client list is daunting: over 4000 IBM-compatible PCs, over 400 Macs, over 700 Unix Workstations, 3 large Vax Clusters, many data acquisition systems, Advanced Computing Lab parallel systems (one with 32 IBM-560s), VM, MVS/XA, MVS/ESA, Convex, Cray X-MP and Cray Y-MP, et. al.

Some of these systems depend on backup and archival processes that are a regular part of the operating system support. Other systems and some PC groups have installed shadow disk systems or other hedges against data loss. Users are pretty much on their own for most of the PCs, Macs and Unix workstations. Some Unix administrators have purchased local cartridge drives, a few with automated mounting, but with the high speed networking we have in place, local tape is not the best solution for speed and probably not for data integrity either.

About 40 workstations are backed up through the network using CFAM. This regular use of the CFAM client commands, has revealed a lot of advantages inherent in the programmable interface approach. System administrators of the various vendor Unix platforms designed scripts which integrate their Unix System Backup tools (e.g., *bru* on SGIs) with the standard CFAM commands.

The CFAM Server only logs pertinent information, which completely identifies clients, owners, file names and bitfile ids, in the case of errors. This makes daily monitoring of off-shift backups very easy. Another inherent advantage is that we know exactly how the client code should function. No bitfiles are worthless due to unexpected or unexplained code conversions or erroneous record deblocking. Client code software levels are checked and requests are rejected if the software level is no longer compatibility with the CFAM Server. If compatible, but below the current level, the log entry is flagged so we can notify the system administrator to download the latest version of the client code.

Since these backups require no human intervention, they make use of the Network, the CFAM Server and the client workstations themselves during off hours. This provides better utilization of all these resources. In addition, the system administrators are freed from tape procurement and storage concerns, and need little or no free disk space to run the backups. More often then not, the workstation doing the backup is a pseudo MSS server, and is getting its *bru* or *tar* input from NFS mounted remote systems, which are usually on the same network segment. The output is piped into *smig* (CFAM's put command) which uses reliable transmission for the more convoluted network path to the CFAM Server.

We chose to build basic Unix commands on top of the programmable interface but a window environment would also be possible. The significant point is that the identical CFAM client command codes, written in C and using RPC, have been readily ported to Sun, SGI, Apollo, HP, DEC-Ultrix, IBM-AIX, Convex, Solbourne, UTS and

NeXT systems. (On the Apollo and HP platform, a public domain version of Sun's RPC had to be installed. The NeXT installation uses the NeXT Unix emulation support.) Transfer rates are not less than 90% of FTP rates to the same systems.

We also envision other program controlled file transfers as more applications migrate to workstations, such as CADAM to ProCADAM. We want to make sure the "corporate database" still remains, with regular updates from the field. Since different databases will have varying frequency of update, security requirements, compression benefits, etc., using an option rich programmable interface looks very attractive.

Also, as standard products evolve, scripts can be changed to take advantage of improvements without the users' knowledge. For example, if DFS can later do a better job of automatically relocating required data, a script with a CFAM command could be changed to use a DFS mount.

Current Centrally Supported MSS Environment

- Ten to fifteen Distributed Storage Servers (DSS)
  - Various workstations (DEC, Sun, SGI, IBM, Solbourne) dedicated to storage functions.
  - NFS (only) access to users for program downloading and additional temporary space. Each system has from ten to twenty gigabytes of disk storage.
  - Systems are placed as close to their principal users (network wise) as possible.
  - File Systems are periodically backed up to Central UniTree System using batch FTP processes.
- Central Epoch System (Approx. 100 GBytes)
  - NFS (only) Access to users for program downloading and additional temporary and permanent space, although recently its value as additional storage has been diminished because most file systems are nearly full.
  - There is no backup of user data.
- Central UniTree File Manager running on a Vax-Ultrix Platform
  - System now in Limited Production. Ten Gigabytes of disk, and one STK Automated Cartridge Library.
  - Access is via FTP for backup of DSS systems and for a limited number of test users.
- Central File Archival/Migration Service (CFAM)
  - In house developed central MSS server running on MVS/XA system using TMS, DFHSM, and sharing two STK Automated Cartridge Library units with native MVS and VM archival/backup uses.
  - Has about eighteen gigabytes of dedicated disk storage and supports direct file transfers to/from ACS cartridges for large files.
  - End user interface is a set of six commands (only five for the Crays) which store, retrieve, test and delete files, and provide catalog listing and user authentication services.
  - Supplies all MSS service for our Cray X-MP and Y-MP users. File transfers use modified Cray-MVS Station software and dedicated channel attached FEIs.
  - Supports networked users, but at about 10% of the Cray throughput (a lot of Ethernet still in the network path). Uses standard RPC Client/Server logic using reliable (TCP) transmission layer. The Network-CFAM commands are enhanced versions of the Cray-CFAM commands, supporting more options and pipes.
  - Currently represents the bulk of our central MSS service with about 120,000 files and 500 Gigabytes of data.
  - Maintenance and migration utilities allow rational file selection. For example, bitfiles can be migrated based on: ownership, size, time since stored, time since last referenced, original source of data, physical volume containing the bitfile, and many of these in combination. There are no magic or arcane ratios. "What if" runs which just identify the selected bitfiles and summarize the resulting disk storage freed are used, if non-standard criteria are needed to deal with exceptional disk cache pressure. The actual migration uses standard MVS DFHSM commands that are batched for efficiency.

Concurrent Services and a Smooth Transition with Two MSS Levels

The Distributed Storage Server level, the MSS closest to the distributed clients, should be the principal support for most of the daily client needs. Treating it as a host for standard network mounted file systems should allow us to move as fast as the industry in implementing DFS as it becomes available. It will also allow us to install high performance NFS servers if that seems to be a better approach. Our Epoch may just appear like another DSS or it may be replaced by a couple of them.

The next layer up the MSS hierarchy will probably include UniTree and CFAM for a while. We expect that disk costs will be one of the deciding factors to move away from MVS, although currently CFAM requires about 1/5th the disk space of UniTree's Central File

Manager, because it supports direct to tape transfer of large files.

New applications, such as local storage support for our users of a remote Cray C-90, would be hosted by UniTree, using UCFM's FTP support. We would also gradually phase out CFAM by redirecting CFAM bitfiles to UniTree but still making CFAM catalog entries. Instead of indicating a physical device, the media field in those entries would be set to UniTree.

Once a significant number of the currently referenced CFAM files exist on UniTree, we would then remove the extra cataloging step to CFAM. Remaining unreferenced CFAM files would be copied to UniTree and the CFAM client calls would be redirected to a CFAM call interceptor running on UniTree. (Note: conversion of our Cray-CFAM users could theoretically be included in this transition, but I/O inefficiencies and excessive use of the MVS CPU for TCP/IP protocol support would probably dictate a specialized conversion.)

Other Scenarios

As always, non technical factors could totally revise our plans, but we also know of three technical variables. First, some of the features currently missing from UniTree are critical for its full production use. Secondly, where OSF-DSF or high performance NFS might fit in needs to be resolved. Lastly, will some vendor deliver a "state-of-the art" data server that integrates MSS back end services?

Integrated Back End

Systems that integrate the back end of MSS into the network access method have the most seamless look to an end user. Unfortunately this can commit you to a single vendor, thus losing all open systems advantages. This would probably only be acceptable if it offered an otherwise unattainable "state of the art" capability. (Within the IBM mainframe MSS world, the proposed STK instantaneous backup on Iceberg might be considered such a capability.) If a programmable interface existed for this product, it could still be integrated into our MSS Complex.

Diminished Value for UniTree

If some of the improvements to UniTree don't happen soon (e.g., better message handling, programmable interface, direct to tape) it would be used internally by the MSS Support Group to backup DSS systems, but direct use by the general user community would not be encouraged. If only the program interface is missing, new applications could use FTP and we would move CFAM files on a request basis for users who wanted all their files on UniTree.

A Middle MSS Layer

Integrating DFS into the Lewis MSS Complex might be done best at a middle layer. In this case, DSS systems might make use of this second layer to improve data availability and delivery. Most of the back end support would then be between the middle and top layer.

Conclusion

Data storage that is not reliable and readily expandable hinders the development and operation of all your site's applications. It also results in the use of extraordinary people resources to deal with its consequences. Once users perceive poor performance due to storage limitations, or worse, cannot depend on the integrity of their data, it is very difficult to establish reasonable policies. Aberrant disk purchases and backups of every file in sight may soon follow.

Mass Storage Systems have the additional requirements of securing, maintaining and delivering the data over months and years, while everything else is in flux. If a site uses MSS services to provide overflow relief for other local storage resources, a fairly large MSS reserve capacity must be maintained. Large high performance disk arrays add yet another complication as do the systems that provide waves of data to them.

It is clear that a substantial growth in MSS capacity and capabilities is overdue. To the extent that a standard can hasten this development—great! To the extent that the standard will be "after the fact" documentation of a de facto standard—good! To the extent that the standard just represents IEEE committee momentum—forget it!

Just as the television industry and FCC have to decide whether, or how much, to embrace High Definition TV or leap frog to newer digital technologies, the IEEE has a dilemma. Should development of the full range of lower-level standards be continued or should some effort be put into standards which address a more comprehensive MSS interface?

Acknowledgment

This paper is based, for the most part, on MSS work done at the NASA Lewis Research Center in Cleveland, Ohio.