Efficient Long-Term Data Storage utilizing Object Abstraction with Content Addressing

Robert Primmer
EMC, Centera Division
Hopkinton, MA, 01748-9103
Primmer_Bob@emc.com

Abstract

HDD prices have declined and reliability has increased to the point where economics no longer dictate that IT settle for less efficient means for long-term data storage This advancement comes at a key point in time as the need for storage not only continues to increase, but the need for immediate access to data likewise increases as technology advances allow the pace of business to quicken. However, inexpensive disk in and of itself solves only the economic problem. New mechanisms are required to increase the level of abstraction for both applications and IT Operations. Absent such abstraction the ability to deal with ever increasing levels of data storage grows arbitrarily complex.

This paper contrasts the traditional hierarchical file system abstraction to an object abstraction. The general model of an object store is later expanded to include a method of storage addressing based upon the data itself. Finally, a specific architecture is presented and contrasted to the general model.

INTRODUCTION

Traditionally IT has had to choose less efficient means of storage for certain classes of data (e.g. long-term data storage) for economic reasons. To this end, legacy systems have used means such as tape or optical to offload data from primary HDD storage for long-term archival. However, such a trade-off becomes something of a faustian bargain, where reduction in purchase price is traded for decreased efficiency attributing to the manual nature of the medium. Today the dramatic reduction in HDD cost [1] coupled with MTTF rates greater than 100 years [2] obviates the need for IT to make do with lesser means of data storage. As these twin dynamics play out over time, HDD will increasingly become the medium of choice for all data storage; removing the need for IT to artificially create separate storage policies for various types of data.

While reduced cost and increased reliability solves the problem of physical medium, significant work remains to improve the lot of both IT Operations (Ops) and the application developer. Long-term storage of data, where time is measured in years to decades, introduces challenges beyond those typically associated with short-term data storage. Specifically: how can applications efficiently and effectively manage the complexity of 1000's of millions (10^9) to millions of millions (10^{12}) of files over extended periods of time.

The continued increase in data creation is well documented. In particular, there is substantial evidence that storage requirements for one category of data is increasing at an increasing rate [5, 6]. This class of data has several monikers, the more prevalent include: fixed content, reference data, and unstructured data. Whatever the name, this class of data is characterized as data that changes either infrequently or not at all; e.g. medical imaging, archived documents and email, etc.

To move forward in these areas requires advancements in software methodology to complement the advances in hardware. This paper looks at means of increasing abstraction from the perspective of the application and IT Ops, as well as presenting a new model for addressing storage based on the content of the data itself.

DATA ABSTRACTION – PRESENT DAY

For over 30 years operating systems have provided a storage abstraction to applications (Figure 1), thereby protecting applications from the details of the physical medium. This abstraction has been critical to the advancement of future development, effectively allowing application programmers to focus on the particulars of their application with little concern or understanding of the inner workings of data storage, buffering, and retrieval [4, 7].

While the kernel does an excellent job of abstracting low level device detail via abstractions such as files and directories, applications (and IT Ops) still own responsibility for locating data and validating its authenticity.

There has not been material advancement in general data abstraction since the introduction of the operating system. For example, today it remains the responsibility of the application to know the precise location of all data files it uses. Often this translates into knowing the fully qualified pathname, including on which host a given file resides. While various methods exist for masking some of this detail, e.g. the use of environment variables as placeholders for file location, ultimately the responsibility resides with the application to correctly locate files.

In the dynamic world of a typical IT environment file location is seldom static. When applications that have worked perfectly well suddenly fail, seasoned developers have

learned to first check for missing files as curative action. The aggregate cost to IT for such a fragile system is substantial. Moving a single file, deleting a single link, or changing the access control on a single file can easily result in hours of downtime.

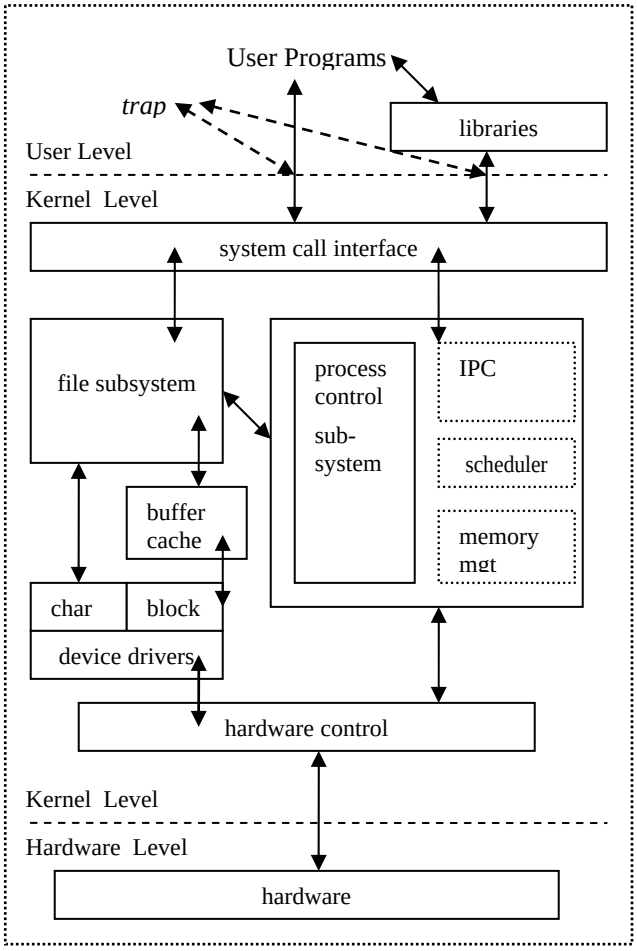

Figure 1. Kernel Abstractions [7]

Hierarchical File Systems (HFS)

Today the most common form of commercial file system is the hierarchical structure. This model works well on several levels. For the user, it represents a readily understood paradigm, providing an intuitive mapping to the familiar file cabinet→drawer→folder→file model prevalent in the paper world. For the operating system it serves as a means of presenting clumps of data as logically discrete units (files, directories, etc.). This has the effect of simplifying both kernel and application code while reducing the time required for operations that traverse a file set.

Models for Large File Dispersal using HFS

Recognizing that as file counts continue to grow the problem of file location grows likewise, storage researchers have proposed a number of models that attempt to mitigate this effect. The majority of the proposed models create some form of central naming authority that has jurisdiction over a domain of storage systems [11-14]. The principal strength of these models is their attempt to retain legacy HFS semantics; predictably, this is also their most common weakness. Adding yet another layer onto the OS/FS model necessarily adds complexity while simultaneously increasing fragility, proving yet again the wisdom of Occam's Razor. Further, bound by the constraint of traditional HSF semantics such systems have difficulty achieving meaningful distinction.

While these models are ambitious in their attempt to normalize a large namespace, they hedge on the fundamental question of who owns responsibility for file location: the application or the file system?

Though hierarchical file systems (HFS) have proven an excellent abstraction as a general purpose file system, further refinement is required to address the unique operating requirements of fixed-content data storage, retrieval, and data validation. Particularly with respect to removing the burden from the application to meet these requirements.

INCREASING ABSTRACTION

A new model has emerged that departs from the standard approach of emulating an expanded HFS over increasingly greater geographies. This model allows applications to treat data as an abstraction: opaque objects that present little to no detail about the underlying data. Rather than a hierarchy of directories and files, this model presents the illusion of a flat object store. Applications submit data of arbitrary type and are returned a unique "handle" to the data that allows later data retrieval without requiring any greater understanding of the underlying mechanisms used to store, protect, validate, and retrieve the data; essentially creating a "write once, read everywhere" method of operation. Details of where or how a given object is stored are self-contained within the object store itself. Therefore, the object store inherits all responsibility for data location, integrity, and immutability. The application is successfully divorced from the business of file location and management.

A Brief History of Abstraction in the Data Center

Computer science has steadily sought to increase abstraction as a principal means of advancement. In the IT space this is evidenced by the move from a monolithic system that performed all functions to an increasing dispersion of function across multiple devices. Individual devices are thus allowed to be less complex while gaining increasing sophistication in the smaller set of tasks for which the device is purpose built.

Initially, all computing resources were stored and performed in a single monolithic structure. All service depended upon access to the central site. A failure with the central computing element resulted in a complete black-out of all service. Further, the complexity of the central computing element was necessarily quite complex as it had to have native intelligence to perform every function (Figure 2).

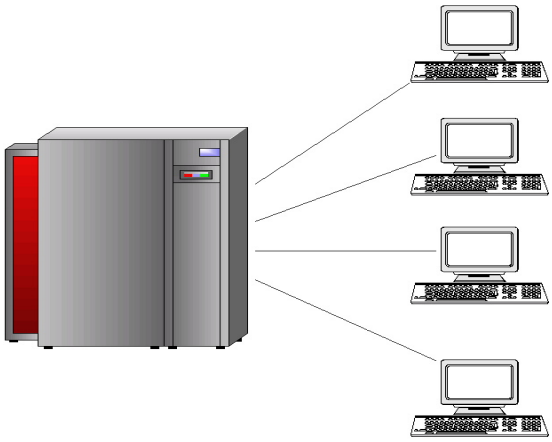

Figure 2. Monolithic Structure

Over time it became evident that there were efficiencies to be gained from separating the *presentation* of data from its *execution*. This lead to a 2-tiered model commonly referred to as client-server, where the client had principal responsibility for the "presentation layer" and the server held principal responsibility for the "execution layer." While this model provided some efficiencies, both in application programming and in purpose-built appliances, there remained a tight coupling between the execution server and the data store, as depicted in Figure 3.

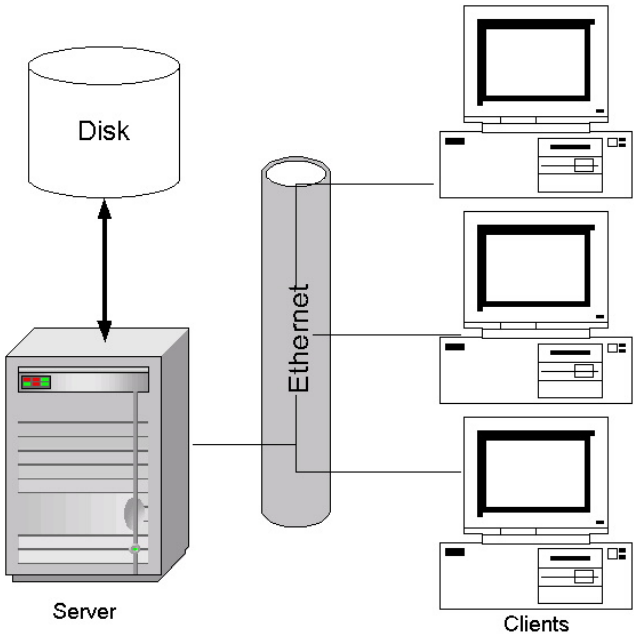

Figure 3. Client / Server Architecture

The next logical stepwise function was to separate the data store from the execution layer. This led to the advancement of the 3-tier (also referred to as the *N*-tiered) model, as depicted in Figure 4.

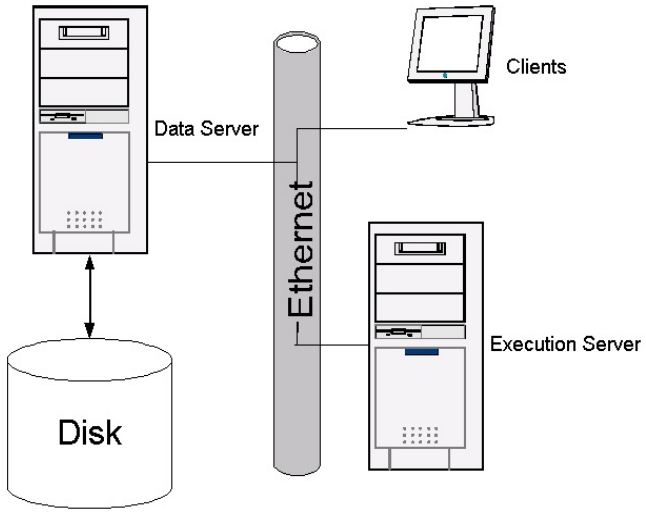

Figure 4. Client / Server / Server Architecture

While architecturally sound in principal, the *N*-tiered model never achieved its full potential in practice as applications operating at the execution layer are not truly separated from the details of the data store. While physically separate, there remains an incestuous relationship between the applications that execute upon the data and the details of how data is stored.

This failure to meet the ideal of the *N*-tiered abstraction model is not simply an academic distinction. To the extent that the application developer, and by extension IT operations, must be concerned about the detail of data, they are distracted from performing the task at hand. For further progress to be made the ideal of *N*-tiered model must be realized. The alternative is relative stagnation, with only marginal gains in application development.

Completing the Ideal

The model of universal object store represents a quantum step forward in the goal to increase abstraction to the point where application developers are all but completely relieved of the details of data storage. While this significantly reduces the burden on the application developer, it has the beneficial side-effect of reducing IT complexity. The exposed and often fragile link between applications and the file(s) they use is broken, thus allowing IT greater flexibility in the organization - and more importantly reorganization - of data. IT Ops benefits as well as this additional layer of abstraction eliminates the block/file distinction for managing data, which causes so many headaches for IT managers and end users [3]. Further, by introducing sophisticated software within the object store that owns all aspects of data storage and its management, this effectively moves "storage" up the value chain, both to the application developer and IT Operations.

Protection from Changes in Physical Mediums

This degree of abstraction provides protection not only in the important area of abstracting file location, but in the area of physical medium as well. It is a given that physical

mediums will continue to change over time. HDD's continue to live on, despite predictions of their demise, and continue to improve density at impressive rates. Nonetheless, it's certain that this curve cannot extend indefinitely; eventually further meaningful density improvements will become impossible. Given the physics of the medium it is estimated that areal density of HDD's will reach maximum possible density at ~150Gb/in^2. At which point the problem of superparamagnetism[1] is expected to prohibit future HDD density increases.

Regardless of precisely *when* it occurs, it is generally acknowledged that it will become necessary to move to storage mediums beyond the 2-dimensional, with work on 3-dimensional light (holograms) as storage medium already begun. Whatever the medium, the opaque nature of the object store protects applications from eventual changes in the medium.

The Payoff

The benefits of abstraction in computer science and electrical engineering have long been understood. Abstraction can reasonably be attributed with providing the foundation necessary for future achievement; both in computer hardware and software. Kernel-based HFS's have provided application developers a sound data abstraction model for more than three decades. However, with the addition of an increasing numbers of file servers dispersed over increasing geographies, file management complexity continues to increase geometrically.

Application developers and IT Operations alike need a refinement of this classic model to effectively deal with massive quantities of data. The advent of commercial-grade object store technology, coupled with dramatic advancements in disk price/performance, makes it possible for IT organizations to keep ahead of the complexity curve in a cost-effective manner.

CENTERA – CONTENT ADDRESSED STORAGE

Centera is a commercial quality object store that combines a sophisticated software "operating environment" (OE) with low cost, highly reliable hardware. The hardware is comprised of a set of discrete nodes coupled together in such a manner that disruption in any one node does not affect the operation of other nodes, yet appears to the application (both physically and logically) as a single unit, or cluster. This architecture, termed RAIN (redundant array of independent nodes), allows for incremental scaling, from terabytes to petabytes, without the need to disrupt system operation. Further, since the system can continue to operate in the presence of failed nodes, service is reduced to a regularly scheduled event where failed nodes are simply replaced with new.

Benefits of an Intrinsic Operating Environment

While the advancements in hardware that make such an architecture not only possible but affordable are truly impressive, the majority of features provided by Centera come from the operating environment, CentraStar. CentraStar presents a flat object store to the application. Applications submit data and are returned a global handle that uniquely refers to the data stored. The handle returned is a function of the data submitted. Upon submission, a 128-bit hash (called the Content Address, or CA) is calculated at the application server. This value is then recalculated upon arrival of the data to the CentraStar server. If the values match the data is stored, otherwise an error is returned indicating a corruption occurred during network transport. Thus the CA serves the dual role of unique object handle and data checksum. The checksum function is important as it serves as the means for integrity checks throughout the life of the data. It's one thing for an application to store mortgage application scans, it's of greater importance that the data retrieved 30 years later can be validated to be an accurate bit-for-bit representation of the data originally stored.

Data stored on Centera is automatically mirrored to multiple nodes. Thus the failure of a node does not result in lost data. Rather, in the event of a node failure all data that resided on that node is automatically regenerated by CentraStar to assure there are always at least two copies of all data objects. All of this occurs without any action by the application or IT Ops. This model extends equally well when replicating data between Centera clusters. Data stored on one Centera will automatically be replicated to backup, again accomplished without human intervention.

All of these actions are able to occur without human intervention or setup due to the intrinsic intelligence that CentraStar brings to the Centera offering.

The Next Step Forward

The combination of opaque object store and unique data handle (CA) provide the needed next step forward in data abstraction. With Centera, applications are successfully abstracted from the detail of data storage and co-ownership of the data management function. IT Operations is likewise abstracted from traditional storage details such as RAID types to choose, LUNs to bind, or file systems to create [8].

This model of storage is named "Content Addressed Storage" or CAS. CAS significantly extends the benefits of the object store contrasted to an HFS in a number of significant areas.

1. CAS creates a self-verifying namespace. With an HFS no such function is provided.
2. The object handle (Content Address) is guaranteed to return the data originally stored in the precise form that

[1] When maximum areal density is reached, the magnetic energy holding the bits in place on the medium becomes equal to the ambient thermal energy within the disk drive itself. When this happens, the bits are no longer held in a reliable state and can "flip," scrambling the data that was previously recorded.

it was originally stored, irrespective of changes to the Centera configuration. HFS provides no such guarantee.

3. Unlike CAS, the HFS pathname holds no intrinsic semantic meaning.
4. With CAS the namespace, as seen by the application, is guaranteed not to change. With HFS namespace changes are not only possible, but frequent.

CONCLUSION

The sum function provided by Centera with CentraStar creates an efficient spinning disk solution to the problem of long-term data storage. With price removed as a barrier the need for legacy models for long-term data storage, e.g. tape or optical, is removed. Now all categories of data can be delivered and accessed in the same manner, with sub-second storage and retrieval irrespective of the length of time the data was stored, or the volume of data stored. Further, the addition of an intelligent operating environment allows for increased abstraction and self-healing/self-managing capabilities that could not be realized with either legacy storage mechanism or with an HFS running on top of spinning disk.

ACKNOWLEDGMENTS

Special thanks to contributors and reviewers of early drafts of this document, specifically: Steve Todd, David Black and Ric Wheeler.

REFERENCES

[1] Morris, R., Truskowski, B., *The evolution of storage systems*, IBM Systems Journal, Vol 42, No 2, 2003

[2] Maxtor, *MaXLIne ATA Hard Disk Drives*, Data Sheet

[3] Villars, R., et al., *Disruptive Innovation in the Enterprise: Content-Aware and Near-Line Storage*, IDC Technology Assessment, June 2003

[4] Vahalia, U., *UNIX Internals: The New Frontiers*, Prentice-Hall, Inc. Upper Saddle River, NJ, 1996

[5] MacFarland, A., Kahn, M., *Retrieving the Needle in the Haystack – EMC's Centera Manages by Content*, The Clipper Group, May 2002

[6] Enterprise Storage Group, *Product Brief – EMC Announces Centera*, May 2002.

[7] Bach, M.J., *The Design of the UNIX Operating System*, Prentice-Hall, Englewood Cliffs, NH, 1986

[8] EMC^2, *EMC Centera – Content Addressed Storage, Product Description Guide*, 2003, located at: http://www.emc.com/pdf/products/centera/centera_guide.pdf

[9] Rothenberg, J., *Ensuring the Longevity of Digital Documents*, Scientific American, 272(1), Jan 1995

[10] Laplante, P., *Dictionary of Computer Science, Engineering and Technology*, Lewis Publishers, Inc., Dec 2000

[11] Brandt, S., et al., *Efficient Metadata Management in Large Distributed Storage Systems*, University of California

[12] SGI, *SGI CXFS: A High-Performance, Milti-OS SAN Filesystem from SGI*, 2002

[13] Menon, J., et al, *IBM Storage Tank – A heterogeneous scalable SAN file system*, IBM Systems Journal, Vol 42, No 2, 2003

[14] Microsoft, *Distributed File system: A logical View of Physical Storage*, 1999

[15] Tharp, A., *File Organization and Processing*, John Wiley & Sons, Inc., 1988

[16] Custer, H., *Inside Windows NT*, Microsoft Press, 1992