Designing an Architecture for a Distributed Statewide Mass Storage System for Video and Other Information Objects

Kevin L. Wohlever - OSC (kevin@osc.edu)
David Barber - OhioLINK (david@ohiolink.edu)
Al Stutz - OSC (al@osc.edu)
Paul Buerger - OSC (paul@osc.edu)

Ohio, like other states, faces the challenges of enhancing its educational infrastructure in order to increase the ability of its educational institutions to collaborate on degree programs and provide distance education to support life long learning. Delivering educational materials from higher education institutions to K-12 schools and to provide a means for collaborative higher education programs has become a goal of Ohio's state government. The adoption of this goal has resulted in the creation of an organization, the Ohio Learning Network (OLN), to facilitate and lead state higher education institutions towards this goal.

Video is the preferred means of achieving these programmatic goals. However, ubiquitous use of video is currently beyond the capability of the state's technological infrastructure; among the other shortcomings of this infrastructure, bandwidth and insufficient storage are particular problems. To respond to these infrastructure issues, OLN has inaugurated a project to build a statewide video Intranet, the OVI (Ohio Video Intranet) Project. It is hoped that by collaborative action between Ohio's statewide technology organizations and higher education institutions a relatively seamless and interoperable video infrastructure can be build that will be able to link higher education institutions with each other and K-12 schools.

Two of Ohio's statewide technology organizations, OhioLINK (Ohio Library and Information Network) and OSC (Ohio Supercomputer Center) have a common interest in solving a problem of the OVI--how to store and distribute video. Ohio Supercomputer Center's High Performance Computing (HPC) Division has been involved in large-scale storage of scientific data since 1987. The Center provides a central mass storage server for five supercomputers on site at OSC. OARnet, also a part of the OSC, is the Internet Service Provider for Ohio's universities and colleges. OARnet is also a member of I2 and provides Ohio with connections to the vBNS and Abilene.

OhioLINK is a state-funded consortium of more than 70 universities and colleges that centrally purchases and manages electronic information for those institutions. OhioLINK also is developing a role as a central repository of multimedia produced by Ohio's universities and colleges. To manage this content OhioLINK relies on OARnet's network and has been working with OSC on linking its computers with the OSC MSS to develop a multi-Terabyte system for storage of multimedia content.

To fulfill their statewide roles and support the development of the OVI, OhioLINK and OSC are developing an architectural design for a distributed statewide storage system (DSSS). By creating a central archive with a large capacity and low cost storage, and combining it with distributed video storage and streaming nodes that are nearer on the network to Ohio higher education institutions, OhioLINK and OSC believe that they provide additional storage for video, limit the burden video would otherwise create on the state's central backbone, and remove the need to stream video from central servers. While the DSSS

is primarily being created to respond to the need for distribution of video, it is intended that it serve as a place for storage of other forms of information objects such as multimedia presentations or high-resolution images. It is both organizations' goal to complete the architectural design for the DSSS and begin implementation of a testbed later this year.

This paper describes the general constraints that have been set on the design of the DSSS architecture. It also describes OSC/OhioLINK's functional requirements for the DSSS. With these constraints and requirements in mind, the paper analyzes three technologies that might be used to construct the DSSS. These three technologies are distributed Internet caches, distributed file systems, and Jini/JavaSpaces. For each technology, an assessment of its ability to meet the functional requirements of the DSSS is presented. Finally, some conclusions are made about the next steps for the DSSS developments given the relative shortcomings and merits of these technologies.

By considering these technologies, the paper evaluates both technologies like caching and distributed file systems with a strong foundation in current practice and a newer (and more unknown) type of distributed system, Jini/JavaSpaces. The consideration of these three technologies also contrasts two solutions that are file based with a distributed object system.

I. CONSTRAINTS AND REQUIREMENTS OF THE DESIGN OF THE DSSS ARCHITECTURE

A. GENERAL CONSTRAINTS

OSC/OhioLINK have set the following constraints on the design of the DSSS:

1. Use of Production Components

First, the components used should not be research systems. While the creation of distributed storage systems for the Internet may be a research project, in order to assure maintainability of the DSSS, the design goal of limited requirements for software development has been adopted.

2. No Changes to Current Network Services

A second constraint on the DSSS architecture is that although changes to DNS are at the core of the proposed I2-DSI architecture, the DSSS will use WWW servers to resolve the URN addresses of information objects. The WWW servers in the DSSS will need to redirect users to other caches through HTTP redirect responses.

This use of WWW based location resolution is required because it is not believed that it would be feasible to get all of the universities and colleges served by OSC/OhioLINK to modify their DNS servers. In addition, by building on the modular structure and open structure of public domain WWW servers, it is believed that more frequent changes can be made to develop the algorithms for resolving the location of cached objects and regulating the caching of information objects.

3. No Changes may be made to User Desktops

As OSC and OhioLINK serve more than 500,000 faculty, staff, and students at more than 70 universities and colleges and also intend to serve Ohio's K-12 schools, it would be impossible to require these users to make any configuration changes to their WWW browsers. Even if the need to do this could be communicated to the users, it would be impossible to provide them with the support they needed. As a result of this factor, it will be impossible to have users modify their browsers to use specific proxies or caches.

4. Initial Use of OhioLINK/OSC Controlled Servers and Content

A fourth constraint on the architecture is that it will initially only incorporate distributed systems that can be centrally controlled by OhioLINK/OSC. Similarly, the storage will initially only serve as a cache of video and other information objects archived in OhioLINK/OSC's central storage system.

In later phases of this project, it is expected that these constraints will be lifted and the architecture will be expanded to include storage systems that may be controlled by individual campuses and include locally produced content. When this expansion occurs, the architecture will need to permit resolution of objects to these additional local stores.

B. FUNCTIONAL REQUIREMENTS

OhioLINK/OSC have also set the following functional requirements for the DSSS architecture:

1. Transparency of Location

The location of an object should be transparent to a DSSS application. The distribution of the object must be transparent. One network identifier must be used for the object wherever it is stored.

2. Object Movement

It must be possible to govern which types of objects in the central mass storage server can be moved to a particular storage server.

3. Object Locality

It must be possible to identify the best location where an information object should be placed and then served to a remote system. Best may have multiple definitions including such factors as smallest number of hops, highest bandwidth between server and remote host, and/or server load.

4. Object Recovery

There must be a way to recover or rebuild the collection of information on a distributed storage system after a system problem.

5. Object Reservation

It must be possible to migrate information objects to a distributed storage server based on demand or on a predetermined schedule. It must be possible to restrict the ability to migrate the object to specific users or user groups.

6. Object Routing

It must be possible to determine the best method for moving data to a storage server from the central OSC/OhioLINK archive where best is again defined as the shortest network distance from the user.

7. Object Security

It must be possible to restrict access to information objects to specific individuals as well as to groups of individuals.

8. Resource Management Policies

It must be possible to set policies for the duration of residence and the deletion of objects stored in the DSSS' distributed storage locations. It must be possible to delete cache resident information objects based on various criteria including frequency of use, file size, and media type.

9. Limited System Administration

The amount of system administration work required by distributed storage servers must be minimized.

10. Addition of Storage Capacity

It must be possible to add new storage devices or servers to the DSSS. The architecture must minimize the complexity of adding new servers.

II. ARCHITECTURAL OPTIONS FOR THE DSSS

The remainder of this paper will describe the architectural approaches that OSC/OhioLINK are considering in order to try to meet the specified functionality requirements and design constraints. In so doing, it continues the outline of issues and problems presented in March of this year by OSC/OhioLINK in a presentation to the Internet 2 Distributed Storage Infrastructure Conference in North Carolina.

A. RESOLUTION OF OBJECT LOCATIONS

We begin by considering the problem of how the DSSS architecture must handle the problem of resolving the location of information objects. The need to use HTTP rather than DNS to resolve the location of information objects has an important effect on the architecture of the DSSS. It means that every request for an object in the DSSS must be sent to a WWW server. Then that WWW server must either redirect the user to a different WWW server or provide them with the object.

The significance of this process for the DSSS architecture is that the functional requirement to have transparency of location will need to be accomplished through the use of applications on WWW servers. DSSS applications on WWW servers will need to be the interface to the underlying mechanism used to manage the files whether that mechanism is a distributed cache, file system, or Jini/JavaSpace. There must be a program (CGI, Servlet, etc.) on each participating WWW server that must interact with the DSSS storage management system. This program will be the mechanism that provides access to the desired information object.

Thus, unlike a traditional cache hierarchy where the client is an individual's browser, the actual client in this case will be the WWW server program. Similarly, in the case of a distributed file system or a Jini/JavaSpace, the DSSS application will act as the client, not software installed at the remote user's workstation.

B. Mechanisms for Object Management

Three distributed storage technologies are being considered as a means of building the DSSS: 1.) Distributed Internet Caches; 2.) Distributed File Systems; and 3.) Jini/Javaspaces. For distributed Internet caching and distributed file systems, their ability to meet the OhioLINK / OSC functional requirements are rated. A rating of 1 to 5 is given with 1 being highest and 5 the lowest. No such rating is given to Jini/JavaSpaces given its novelty. These evaluations describe the ability of each technology to provide the DSSS WWW application client with an environment that meets the functional specifications. The ideal storage management mechanism for the DSSS would meet all of the functionality requirements so that the capability to perform the function does not need to be developed as part of the DSSS WWW application.

1. Caching

The goal of the DSSS to have a distributed set of servers where information objects are cached as needed is very similar to the goal of previous projects that developed distributed Internet caches, e.g. the Harvest project. Because of this similarity, it is reasonable to consider the use of Internet caching technologies to construct the DSSS.

Description

There are a number of protocols under development that are designed to enable the management of distributed Internet caches. The primary protocol currently in widespread use is the Internet Cache Protocol (ICP). As a result, it will be the ICP protocol and ICP-based caches like

Squid that will be considered here as the potential means of constructing the DSSS.

The goal of a hierarchical Internet caching system is to reduce the use of network bandwidth to request objects from remote servers. Instead of requesting an object from a distant server, a request for an object is sent to a nearby cache. If the cache does not have desired object, it will obtain it. Then, the next time any cache user wants to access that object they can do so by obtaining it from a point in their local network. It is not necessary to obtain the object from its original source and consume network bandwidth on the network links to the origin server.

Hierarchies of these caches provide a series of caches from which the information object can be obtained. A cache may be a child of a parent cache. This parent may have other child caches. A parent may be a child of another cache. When a local cache does not have an object, it will contact its parent. If that parent has the object, it will provide it to the child. ICP is the protocol used to manage the interaction between these collections of caches.

Meeting the Functional Requirements

Transparency of Location	5	Application must contact other cache servers to find an object.
Object Movement	3	Control over object caching might be achieved if cache obeys Cache-Control headers in HTTP.
Object Locality	2	ICP reports to the server the time it took to contact another server.
Object Recovery	5	No support. In the event of a disk failure without backup, caches wait for a new request for an object before adding it to the cache.
Object Reservation	4	Most object movement occurs on demand. Limited, if any, support appears to exist for pre-fetching.
Object Routing	2	Each cache uses a predetermined set of parents or siblings from whom it tries to obtain an object. Given the availability of two sources from which an object can be contained, caches obtain the object from the nearest cache.
Object Security	5	No built-in security mechanisms are available.
Resource Management Policies	5	One standard algorithm, LRU (Least Recently Used), appears to be used by most caches.
Limited System Administration	3	Products that do Internet caching are widely used. As a result, limited new knowledge is required of the administrator. However, it would be necessary to appropriately configure the cache to participate in the hierarchy and to install software that would make the cache part of the DSSS.
Addition of Storage Capacity	1	Addition of the devices is as easy as the operating system makes it. Addition of new servers would require implementation of the appropriate Internet cache software and the DSSS application.

While Internet caches and their associated protocol ICP do provide means of organizing the relationships among caches and determining the source of an object nearest to a server, this technology does not appear to provide strong support for policy based controls over object access or movement. It would appear that significant application development would be required

by OSC/OhioLINK to develop a cache with these characteristics.[1]

2. Distributed File Systems

There are three types of distributed file systems that might be used to manage the files in the DSSS: NFS (Network File System), DFS (Distributed File System), and HPSS (High-Performance Storage System). The merits and implications of using each of these file systems are considered separately.

Network File System

Description

The Network File System (NFS) was developed by Sun Microsystems in the mid-1980's. It is difficult to find a UNIX system that does not support NFS. NFS has become a de facto standard for distributed file systems.

NFS is an example of a noncentralized distributed file system. Any machine on a NFS network can be a server; any machine can be a client. To be an NFS server, a node must announce (export) the names of the directories it is willing to make available to clients. To be an NFS client, a machine must attach (mount) a remote directory.

Before anything useful can be done with NFS, a system administrator has to do a fair amount of setup. This "setup" actually becomes ongoing administration. Files are served from one or many systems, each requiring someone to "export" the appropriate filesystems.

On the client side, file systems must be mounted as needed. The information of where a file resides must be made available to each client.

Meeting the Functional Requirements

Transparency of Location	4	NFS does not offer transparency of location. All servers that have the object must be known by the client.
Object Movement	5	Everything is cached by the filesystem.
Object Locality	4	Object locality is not possible beyond using fastest responder for a filesystem at mount time.
Object Recovery	4	Local copies have to be recalled from a server.
Object Reservation	4	DSSS application would need to make explicit local copies of objects. No direct NFS support.
Object Routing	4	Fastest initial responder for the filesystem mount is the best possible scenario under NFS.
Object Security	3	UID and GID are the only methods of controlling access to a file or directory. This requires administrative work to configure. Especially when root access to the client nodes is restricted.
Resource Management Policies	5	Not possible.
Limited System Administration	3	This depends on the configuration complexity. We expect that the administrative workload will be significant.

[1] Because of the similarity between the concept of a series of distributed object caches where the user contacts a central site and is redirected to a distributed cache to the technology developed by Akamai, OSC/OhioLINK are investigating the possibility of using Akamai servers to provide a first step towards development of the DSSS. While Akamai would not provide all the means desired for policy based storage controls or support for all the required types of streaming, it would provide an immediate means of distributing content around Ohio and redirecting users to the nearest cache. It would provide an interim solution while OSC/OhioLINK could construct a DSSS that more closely met the functional requirements.

Addition of Storage Capacity	1	Depends on the NFS server. It should be possible to have easy storage capacity increases.

NFS could be a part of a distributed statewide architecture. Using NFS would require a static environment, with high administrative overhead and with high bandwidth between the clients and servers.

Distributed File System (DFS)

Description

The Distributed File System (DFS) has yet to prove itself as a major player in the distributed file system market. It is important to look at DFS because it was designed from the beginning to act as a worldwide distributed files system.

The primary goal of DFS is to create a worldwide file system that allows any users, security permitting, to access any remote file just as easily as a local file.

DFS is not supported on all platforms and the DCE/DFS software may lag behind OS releases on the systems on which it is supported.

DFS is a non-centralized distributed file system. Any node in a DCE cell can be a server and any node can be a client.

Under DFS, there is no mounting of file systems. The system administrator does not have to set up mount tables, only a DCE cell.

Meeting the Functional Requirements

Transparency of Location	3	The application must still know the fully qualified file name. Although it is fairly easy to access a file on the local cell or a remote cell
Object Movement	5	Everything is cached by the filesystem.
Object Locality	4	Not directly supported under DFS. A good administrator may be able to craft something to simulate the function.
Object Recovery	3	Local copies have to be recalled from a server.
Object Reservation	3	DSSS application will need to take action to cause object to be placed in local cache.
Object Routing	3	No routing support. The path between the server(s) and the client will be used.
Object Security	2	DFS uses DCE Security Services, which include Access Control Lists (ACLs). This expands the UNIX-style file permissions and gives better object security.
Resource Management Policies	4	DFS really does not support much in the way of migration policies. The size and volatility of the disk cache will have the most impact of the migration policy under DFS.
Limited System Administration	4	DFS requires DCE and Kerberos software support for each system that is part of the cell. This can be fairly hefty.
Addition of Storage Capacity	2	It depends on the server being used. DCE administrative overhead.

DFS seems like a logical selection for a distributed statewide architecture. Using DFS would require increased fees for DCE / DFS licensing; plus a large administrative workload. This is not quite the environment that OhioLINK and OSC were trying to foster. The training requirements would be substantial.

High Performance Storage System

Description

High Performance Storage System (HPSS) is a high-end Hierarchical Storage

Management system and archive. It is an open system product that is also an IBM service offering. It is the result of a collaborative project between the US DOE labs and IBM. It is currently support on IBM systems and soon will be supported on Sun systems.

HPSS is network centered architecture, but that is normally thought to be a local area network. Its use on a wide area network is just now being explored.

The various components of HPSS can be replicated and moved about a network. The implications of having multiple servers on a wide area network are still unclear. Synchronization issues must be addressed.

The intriguing function of HPSS is the type clients that can be supported. These clients include client API, FTP, NFS MIP-IO and DFS. It is expected that a global file system could also be supported.

Meeting the Functional Requirements

How well HPSS meets the functional requirements will depend on the system design and the capabilities of the clients. The scoring in this section is purely conjecture, as we have limited exposure to HPSS and the potential configuration opportunities.

Transparency of Location	3	The file access will trigger a request. A name server will begin the process of delivering the file to a client.
Object Movement	3	Data movers offer the ability to control object movement.
Object Locality	4	Depending on the data mover, object locality support may be possible.
Object Recovery	3	Can use capabilities of the HPSS server and data mover.
Object Reservation	2	Object movement can be controlled. It should be easy to develop a tool to do this.
Object Routing	4	This could possibly be a function of the HPSS movers.
Object Security	2	HPSS uses DCE Security Services, which include Access Control Lists (ACLs). This expands the UNIX-style file permissions and gives better object security.
Resource Management Policies	3	A tool should be fairly easy to develop.
Limited System Administration	5	We believe the overhead and support of HPSS and the multiple clients could be extensive. No expertise in this area.
Addition of Storage Capacity	2	This depends on the server and the clients, but should not be a problem.

HPSS with the appropriate clients has the most flexibility to support the functionality that OhioLINK and OSC want in a statewide network. There are drawbacks in cost and support. Especially when merged with DCE and DFS, which will import all the costs and issues identified in that area.

3. JINI/JavaSpaces

Jini/JavaSpaces appears to give considerable flexibility to OSC/OhioLINK to meet the DSSS functional requirements. It would become possible to incorporate much of the DSSS policy infrastructure within software objects. There might be more flexibility for constructing the DSSS than if the system had to use existing cache software or conform to the interfaces and file management characteristics of a distributed file system. The object's properties and methods could be customized to function in whatever way OSC/OhioLINK might desire. For this reason, constructing the DSSS based on system of mobile information objects appears to be attractive. While this alternative appears to offer considerable flexibility and freedom, it does violate one of the DSSS projects fundamental design constraints: to use off the shelf software.

Description

Jini is a distributed computing technology recently developed by Sun Microsystems. It provides the means for distributing network services based on the Java programming language. Jini provides the means for clients to find servers that offer particular services. It also provides the means to easily plug new services into a network, for example a storage server could be plugged into a network, announcing its availability to clients through a Jini Lookup service. A JavaSpace is a particular kind of Jini service that provides persistent storage for objects.

Meeting the Functional Requirements

How well Jini/JavaSpaces will meet the functional requirements will depend on the system design and the capabilities of the clients. Because of the flexibility of Jini and the ability that exists to create Java objects implementing any required functionality, the ability of Jini/JavaSpaces to meet OSC/OhioLINK's functional requirements is not rated. Instead, ideas are presented on how Jini/JavaSpaces could be used to accomplish those requirements.

Transparency of Location	Jini Lookup services may be able to hide object location.
Object Movement	No direct support of a movement policy is available, but a client front-end program could be built.
Object Locality	Through the use of Jini Discovery and Lookup services, it should be possible to construct a service to do this.
Object Recovery	Only in the event of a disk failure, without a backup would it be necessary to rebuild a JavaSpace. The JavaSpace is persistent.
Object Reservation	No direct support of a reservation policy is available, but a client front end program could be built.
Object Routing	An application would need to be constructed which knew about available JavaSpaces and could find the needed object in those spaces based on the use of Jini Lookup services. The application could analyze the time taken to receive a response from each JavaSpace.
Object Security	Security controls would need to be implemented within the object.
Resource Management Policies	JavaSpaces primarily use the concept of a timed lease to limit the duration of an object's residency. It is unclear how an application would be constructed that considered other factors.
Limited System Administration	While implementation of the basic Jini services might be straight forward, implementation and support of DSSS applications could administratively intensive.
Addition of Storage Capacity	New storage devices could be plugged into the network. New servers could also be plugged in and made part of Jini community.

Jini/JavaSpaces are a fairly new technology. Documentation on the technology is only just now appearing. Likewise use of the technology is not widespread, so experience with Jini/JavaSpaces is not readily available. For all of these reasons, attempts to plan applications using Jini/JavaSpaces are necessarily speculative. It would be necessary to initiate a research project to develop a Jini-based DSSS, or to wait until other Jini users accumulate and publish the results of their experience, before it can be determined whether the DSSS could successfully be constructed in this way. These are problems in addition to the more fundamental issue of whether the technology will succeed and become widespread.

III. CONCLUSIONS

Through the course of this analysis of the potential technologies that could be used to construct the DSSS, none appears to be a means through which such a system can easily be constructed. The DFS and HPSS distributed file systems appear to be the best immediate means by which to distribute content objects among a series of distributed DSSS information stores. However, if the longer-term goal of making it easy to add new caches is maintained, these systems present some problems. They do not make it easy or inexpensive for a system administrator at an Ohio school to implement a new node in the storage network.

Given the need to develop a front-end application that integrates the DSSS with the WWW and manipulates the underlying storage management system to implement OSC/OhioLINK's desired policy based storage administration, the best choice may be to develop the application based on Jini/JavaSpaces. Java software can be run on many platforms. Java information objects could also be given whatever custom security and storage policy attributes that OSC/OhioLINK desire. Implementation of Jini/JavaSpaces software could also be easier for most system administrators than implementing a DCE cell or a HPSS server although support of a custom Jini/JavaSpaces application might be a burden on OSC/OhioLINK staff.

Further research into Jini/JavaSpaces will need to be conducted before final decisions can be made about the DSSS architecture.

REFERENCES

[1] Ken Arnold, Bryan O'Sullivan, Robert W. Scheifler, Jim Waldo, and Ann Wollrath.. The Jini Specification. Addison-Wesley, 1999.

[2] W. Keith Edwards. Core Jini. Prentice Hall, 1999.

[3] Eric Freeman, Susanne Hupfer and Ken Arnold. JavaSpaces: Principles, Patterns, and Practice. Addison-Wesley, 1999.

[4] Ari Luotonen. Web Proxy Servers. Prentice Hall, 1998.

[5] Evi Nemeth, Garth Snyder, Scott Seebass, et. al. Unix System Administration Handbook. 2nd ed. Prentice Hall, 1995.

[6] Barry Rosenberg and Johnson M. Hart. Client/Server Computing for Technical Professionals. Addison-Wesley, 1995.

[7] Ward Rosenberry, David Kenney and Gerry Fisher. Understanding DCE. O'Reilly & Associates, 1993.