Portable Representation of Internet Content Channels in I2-DSI

Micah Beck and Terry Moore
Innovative Computing Laboratory
Dept. of Computer Science
Univ. of Tennessee, Knoxville
{mbeck,tmoore}@cs.utk.edu

Bert J. Dempsey
School of Info & Library Science
Univ. of North Carolina, Chapel Hill
bert@ils.unc.edu

Rajeev Chawla
Sun Microsystems, Inc.[1]
rxc@netscape.net

Abstract

In this paper we explore the issues involved in creating a content model for the Internet2 Distributed Storage Infrastructure (I2-DSI), a scalable system of servers that provides storage-based, content-oriented services on the Internet2 research networks. The key questions surround the construction of a portable, scalable software infrastructure that will support replicated content channels in a highly heterogeneous network. This software infrastructure and its associated content model must be developed in the context of I2-DSI's two main architectural concepts: replication of servers to provide identical services and transparent resolution of client requests to the most proximate replica. We set the stage for our discussion of portable content channels by presented a unified account of replication and resolution in Internet information systems that encompasses caching and replicated servers. Against this background we present the issues of content portability raised by trying to replicate servers in heterogeneous environments, describe a deliberately limited strategy that will allow us to deploy some initial channels quickly, and discuss the requirements that more robust models must meet to achieve a solid, long-term solution for I2-DSI.

1. Introduction

The Internet2 Distributed Storage Infrastructure (I2-DSI) is a scalable system of servers providing storage-based services throughout the Internet2 research networks [1]. These servers are being used to offer *transparently replicated services*: services which are implemented identically at a set of servers rather than just at one, but without the client having to change any standard protocols or APIs. I2-DSI uses replication as a means to achieve high performance in the delivery of services. There are several motivations for this strategy, including:

- The need for communication between client and server to traverse the network backbone is a barrier to high performance when the backbone is congested.
- Class or quality of service can sometimes be guaranteed if client and server reside on a homogeneous local area network.
- Hotspots in network traffic and server load are avoided when geographically distributed clients are directed to proximal servers.

As we will discuss in Section 3, issues immediately arise in the construction of a portable, scalable software infrastructure which will allow us to provide replicated services in a highly heterogeneous network such as the commodity Internet or Internet2. Our approach to addressing these issues of portability and scalability within the I2-DSI project is the main topic of this paper.

In order to set the stage for a discussion of standards content and service replication across I2-DSI we need to be clear about the project's controlling ideas:

1. We need to understand the meaning of and motivation behind "replication" as it applies to Internet content channels, especially as it compares to and contrasts with Web caching (Sections 2.1 and 2.2.1), and
2. We need to be clear about the issues and choices surrounding the mechanism for designating and accessing the most proximate replica from which to receive services (Section 2.2.2 and 2.2.3).

Replication and Resolution — The transparent replication of services complicates the standard client/server model by introducing two new notions:

- **Replication.** A set of *replica servers* must provide an identical service.
- **Transparent Resolution.** Each client request must be directed to a particular server without change to the client interface or API.

The strategy of using replication and resolution to optimize the transfer of information in the Web is not

[1] The author now works at Healtheon Corporation.

unique to I2-DSI. Several approaches to resolution have been implemented within the context of the commodity Internet, but they are restricted by the need for compatibility with other software elements that they cannot control:

- Web caching, which we view as a form of replication, is implemented at the client proxy interface and through various compromises and approximations is able to operate independently of servers (see Section 2.1).
- IBM's eNetwork Dispatcher and Cisco's Distributed Director implement resolution in the authoritative DNS server and so hides replication from client software which uses DNS resolution appropriately (see Section 2.2.2).

I2-DSI is unique in that it is being deployed in the Internet2 community, where we can suggest modifications to both client and server software. It shares with Web caching the requirement of an open, scalable solution that can be implemented across institutional and national boundaries.

2. Proxy vs. Server, Cache vs. Replica

Our view of replication is somewhat broader than is commonly taken. In this section we characterize a number of quite different schemes as examples of replication and resolution, and seek to understand the range of possible architectural choices and their implications. Our taxonomy includes not only proxy caches and replicated servers as currently implemented, but also possible mixed strategies. However, it should be noted that these choices are largely independent of one another, and mixed strategies might yield solutions that are well adapted to new applications.

For instance, a proxy cache intercepts and replies to every request generated by a client, making transparent redirection easy. It obtains local replicas of content by using the standard client interface to HTTP servers. However these choices are independent: a proxy might choose to perform explicit resolution to find a proximal server rather than, or in addition to, consulting its own cache.

The main approaches to replication currently deployed for replication of content are

- the HTTP and ICP interfaces as used by proxy caches, and
- explicit replication of source files through shared file systems or special-purpose replication protocols.

The main approaches to replication policy are

- dynamic replication through caching, and
- static replication policy expressed either explicitly through channel metadata formats (see Section 3.2.1) or through implicitly through management of source replication.

The main approaches to resolution of requests to a proximal replica are:

- consulting the contents of a local cache and neighbor caches,
- relying on the implement of shortest path routing,
- DNS resolution augmented to reflect host proximity, and advanced URI resolution.

2.1 Proxy Servers.

In this common approach, a proxy server interposes itself between a community of Web clients and *origin servers* that fulfill service requests. Proxies were originally developed to propagate Web requests transparently through firewalls that blocked direct TCP/IP connections to the wide area network. However, because a proxy server acts as an intermediary for every client service request, it can do more than simply propagate requests. The Harvest project [2] used a proxy to perform caching of origin server replies and to fulfill requests directly from the cache, while striving to maintain transparency. Today, proxies are used to implement a number of Web after-market services, and some, such as the filtering of requests of requests or stripping away banner advertising, modify the semantics of Web requests.

When using a proxy cache, replication is achieved by maintaining a copy of replies from the origin server. When caches are combined into hierarchies or meshes using ICP (Internet Cache Protocol [3]), replication is furthered by sharing these replies between proxies. A form of resolution then occurs whenever a client request is intercepted: the proxy checks whether it or a neighboring cache has a copy of a reply which it considers valid, and if so that reply is retrieved and returned to the client.

Proxy servers fit into the current Internet structure because they require no changes to servers or to clients. A proxy server can be installed on any network, and it can cache replies generated by any standards-compliant Web server.

A proxy is introduced transparently into the Web client/server protocols either through explicit configuration of client software (e.g. Web browser or FTP client) or through interception of client requests by a router or a level 4 switch and redirection to a proxy (transparent caching). While transparent caching has

gained commercial acceptance, it is criticized by some within the Internet engineering community because it requires the proxy to actually spoof the origin server's IP address, leading to potential problems if security or monitoring tools attempt to interpret the traffic [4].

A major limitation on caches is that not all services can (or should) be cached by any particular Web cache, and some cannot be cached by any. An example of a class of uncacheable services are those which are dynamic, changing on every request. Caches are usually limited to a small number of protocols such as FTP and HTTP, and extension to very different protocols such as those used by streaming media, while possible can be difficult. Because of this, such extensions are usually made only to commercial caches such as Inktomi's Traffic Server [5].

When used with a typical policy of dynamic loading and replacement, proxy caches can reduce the demand for wide area bandwidth by 50% or more, depending on the characteristics of Web traffic [6]. While this is useful in containing costs, it is of limited usefulness in implementing high bandwidth services such as video delivery, which require *quality of service guarantees.* In fact, dynamic caching is most useful in statistically improving the delivery of high demand content.

In order to improve the delivery of high bandwidth content more deterministically, it is possible to use a static, or policy-based, replication scheme. The pre-loading of cache according to a static *channel definition* has been proposed in a number of push schemes [7, 8]. Note that while the term *channel* is sometimes taken to refer to cache preloading metadata described in Microsoft's Channel Definition Format (CDF), we use it to refer to any description of content which can be used for replication.

> *A **channel** is a collection of content that can be transparently delivered to end user communities at a chosen cost/performance point through a flexible, policy-based application of resources* [1].

2.2 Replicated Servers

In this approach, replica servers offer identical services and client requests are resolved to one copy. The chief advantage of replicated servers over proxy servers is flexibility: a replicated service need not be cachable. For example, dynamic content can be replicated by consistently replicating its source files (programs and data) as long as identical execution is possible on replica servers. Similarly, streaming media source files can be replicated as long as software exists on each replica to interpret those files. Replication trades off the problem of cachability for the problems of consistent replication and portability, leaving the problem of resolution to be dealt with.

In our view, proxy caches are a form of replication, but they are different from replicated servers in that caches are loaded by keeping a copy of the reply to a service request rather than through some explicit replication mechanism. When proxies and replicated servers are used together, they generally run on the same hardware platform but do not interact (e.g. preloaded proxies and gaming servers used to serve the home market). While there is no reason in principle why a proxy driven by both dynamic cache and static preloading policy should not be used along with replication of source files in a single software agent, historically they have not been.

2.2.1 Consistent Replication across Replica Servers

While perfect synchronization of replicated files across unpredictable networks such as the Internet is not possible, practical solutions exist which assume stronger communication models. While the commodity Internet does not meet these stronger models, the failure modes of these solutions are considered acceptable for many applications. Advanced networks such as those deployed among the Internet2 member institutions will initially be much less congested than the commodity Internet, and so will exhibit better reliability; in the long run, they will implement Quality of Service connections which give guarantees even in the face of congestion. Thus, the advanced networks being deployed among the Internet2 member institutions come closer to meeting the stronger models assumed by existing replication mechanisms. We plan to use I2-DSI to gain practical experience of the failure modes these replication mechanisms exhibit and their usefulness to Internet2 applications.

Distributed file system (DFS) or distributed database solutions (DDS) offer strong concurrency controls along with other data management functions that protect data integrity [9]. Mature DFS and DDS systems provide replication mechanisms that support creation and management of multiple read-only data volumes or database regions. Directory-based replication such as that supported by the Lightweight Directory Access Protocol (LDAP) standards can be thought of as a limited form of DDS [10].

While technically attractive, the power of DFS and DDS solutions comes at a significant cost in terms of ease of deployment, administration, and configuration overhead. DFS systems have complexity related to file name mapping and transparent sharing across systems. DDS systems introduce the challenge of mapping user

content into database objects. For I2-DSI, an additional parameter is how DFS and DDS solutions, which have generally been deployed inside enterprise-wide networks, will perform when using wide-area networks where packet loss rates are potentially greater and latency higher than traditional deployment environments.

Both DFS and DDS provide powerful, but heavyweight replication mechanisms for strong consistency semantics that many I2-DSI applications will not require. Thus, while I2-DSI will experiment with DFS-based replication, we will also pursue development of lightweight mechanisms for updating weakly consistent replicas

For example, many FTP sites on the WWW are controlled by centralizing updates at a master site and then using a file synchronization utility for propagating updates to the remote mirrors. A popular open-source tool, *rsync*, employs a novel checksumming algorithm for differential file update, i.e., only the changed blocks within a modified file are transmitted over the network [11]. In I2-DSI we plan to experiment with *rsync* since it is a readily deployable, lightweight, cross-platform, and efficient method for synchronizing file systems.

Moreover, we are modifying the point-to-point *rsync* tool for use with multicast protocols. Our derivative of rsync, which we call rsync+, enables a mode in which information exchanged during file system synchronization---inode-level file status information as well as the changed portions of updated files---can be captured in a local file during a point-to-point *rsync* session. This local file is then be distributed to remote replication servers where it is used as input to rsync+ in order to update the local file system replica [12]. This new batch-mode operation for *rsync* trades off local processing for efficient use of network bandwidth, regardless of the network transport solution. However, significant efficiency and performance gains are possible if reliable multicast transports are employed to move the update file.

For the multiple-site replication of large data sets that will be supported in I2-DSI, emerging reliable multicast protocols are very attractive for crafting scalable replication mechanisms. Several transport-layer reliable multicast implementations exist, and commercial products are now successfully deployed in private networks (e.g., http://www.starburstcom.com/). For public networks, a crucial missing element is the lack of standards for TCP-like congestion control algorithms, but encouraging progress is being made by leading protocol designers and researchers in the IRTF to define standards [13]. In high-speed experimental backbones, using non-adaptive flow control or, alternatively, application-level tunneling across TCP connections will be acceptable work-arounds for the short-term [14]. I2-DSI will deploy and evaluate reliable multicast schemes in the context of moving update files generated with our *rsync+* utility.

2.2.2 Transparent Resolution: DNS resolution with proximity

Resolution is a process of name-to-address translation, and in the DNS (Domain Name Service) the Internet has such a mechanism universally deployed. By adding to the semantics of the DNS without changing the interface, it is possible to achieve a solution which requires only appropriate configuration of DNS software but no changes to the client. There are two flavors of DNS with proximity: server side and client side. Both start with the assumption that the channel to be replicated is specified by a single DNS domain.

- Using server side DNS with proximity, when a modified authoritative name server for the domain representing the channel receives a request, it sends the requesting IP address to each replica server, which returns a metric characterizing its proximity to the client. Only the IP address of the server that is closest to the client is returned by the authoritative server. Because of the use of the client's IP address and the volatility of the response, recursive DNS lookup must not be used, and the result is returned with a time to live of zero. The servers may have difficulty determining a distance metric for clients behind firewalls or routers which block certain packet types. Although some efficiency and fault tolerance is lost, this solution does not modify the semantics of the DNS interface, and so it is compatible with all client software. Server side DNS with proximity is used in commercial solutions such as Cisco's Distributed Director [15] and IBM's eNetwork Dispatcher [16].
- Using client side DNS with proximity, all of the IP addresses of the replica hosts are entered into the DNS statically and advertised like any other domain. A caching DNS server or proxy close to the client examines all lookups, and when it detects multiple IP addresses it computes a proximity metric to each server and orders the response by this metric. It is common practice for clients to discard all but the first in a list of returned IP addresses (e.g. this is the usual implementation of the gethostbyname() library call) and this has the effect of returning the closest replica server. Clients which use multiple IP addresses in the order provided to achieve fault tolerance are

likewise compatible with this solution. However clients which reorder the returned IP addresses will not be compatible, and a compatibility mode forcing only one IP address to be returned must be used. Client side DNS with proximity is implemented in sonarDNS, developed by Martin Swany at the University of Tennessee at Knoxville's Innovative Computing Laboratory [17].

2.2.3 Transparent Resolution: Advanced URI resolution

Compatibility with existing interfaces imposes substantial limitations on the power and generality of the approaches we have discussed to resolution. The DNS can resolve only collections of objects named by a single domain name, and has no explicit parameters to control that resolution. Like routing, it was not designed to support more flexible resolution. Various efforts are underway to develop more advanced naming and resolution, including URN project under the auspices of the IETF [18] and CRNI's Handle system[19].

3. Portability of Content in Replicated Servers

`As we discussed in Section 2.2, the use of replicated servers eliminates the problem of services which cannot be cached but introduces several issues: consistent replication, transparent resolution, and portability of content. We have discussed the first two issues, and indicated possible approaches, both short and long term. This section is devoted to issues of portability.

3.1 Portability and Replication

Whenever we have replication of services, the *source objects* for that service must be replicated among the replica servers. Different servers can have quite different models for these source objects. For example,

1. The source model for HTTP servers is quite diverse, and includes HTML files, Java class files, and executable CGI programs.
2. The source model for Web caches and desktop push mechanisms is MIME encoded digital objects as delivered by HTTP and ICP.
3. The source model for streaming media servers is compressed audio and video files and associated synchronization and delivery metadata.
4. The source model for FTP servers is binary files in a standard directory hierarchy.
5. The source model for many highly dynamic services is a proprietary database.

As illustrated in Figure 1, Web proxies are loaded by intercepting responses to Web service requests which are intended to be received by a standards-compliant HTTP or FTP client, their "source files" are the responses specified by those protocols. The source files of a non-replicated server never appear on the network, and there are no restrictions on their format as long as the server software can interpret them. A replicated server that is loaded by source file replication can similarly use any kind of source files as long as they can be uniformly interpreted by the replicated servers.

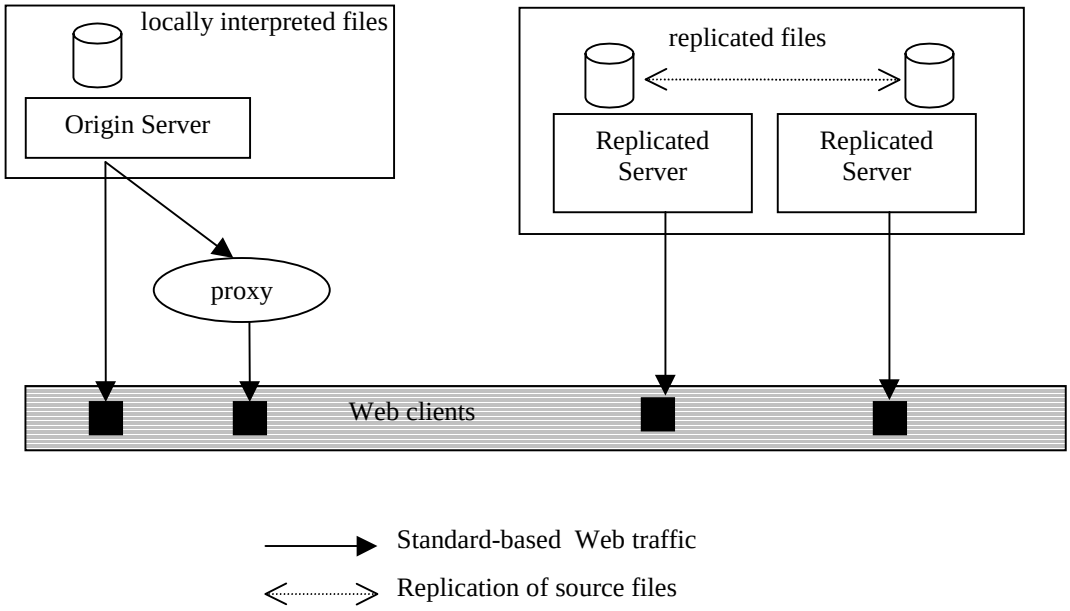

Figure 1: Source models in Web proxies vs. replicated servers

Some de facto standards exist for FTP and HTML based on common file system models and the HTML source language, but the use of even the simplest features beyond the least common denominator leads find a lack of standards which defeats interoperability. Any Web manager who has ported a site between servers is familiar with these problems, some common examples of which are:

- Inconsistent file naming conventions and semantics (index vs. default; .html vs. .htm)
- Server-side includes are completely server-specific.
- Compiled executables are non-portable and different versions of interpreted languages are not interoperable.
- Many database interfaces are proprietary.

The lack of standards has resulted in replication which either is restricted to the common subset of server features or else requires identical server platforms. These restrictions are not acceptable for I2-DSI.

3.1.1 An Initial Model

The strategy of I2-DSI is to deploy replicated services as quickly as possible to obtain experience with their use, to obtain performance measurements and to stimulate interest and cultivate understanding of the project within the application community. For this reason we cannot wait for a portable channel content model to be developed before proceeding. Instead, we will adopt a dual strategy of restricting the services implemented to those that have the most broadly accepted content models and limiting the heterogeneity of our servers. In addition, application developers are warned that the content model may change, and that while the initial model will continue to be supported on the initial servers, additional servers may only support later, more portable content standards.

The initial model will consist of:

- A standard hierarchical file system supporting the intersection of Unix and Windows NT file system semantics.
- Standard HTML without server side includes and with specified server restrictions and conventions including
  - standardized file naming
  - no password protection
- Active content in the form of
  - PERL scripts adhering to a specified API and other conventions, or
  - JAVA using only specified APIs.
- Proprietary streaming files of specified types and versions.

This initial model will be well documented along with rationale and approaches to implementing it as widely as possible. Channel metadata will not be made explicit but will be expressed through manipulation of replication mechanisms.

3.2 Requirements of Portable Channel Representation

The initial model of channel representation described in Section 3.1 is very simple: a channel is a set of files embedded in a hierarchical namespace, chosen from of a small set of file types which are indicated by a combination of location in the namespace and name extension. For example, the file `/usr/apache-docs/foo.htm` is an HTML document which can be interpreted by the Apache server, but the file `/usr/real/bar.ram` is a file in a proprietary format which must be interpreted by the RealMedia server.

This simple solution reflects the limitations of the current content model. The use of the directory structure to infer the choice of server is necessary because different servers implement different dialects of standard file types. A strong type system would allow any standards-compliant server to interpret a file of a given type and would make the location of the file immaterial. However, the choice of server also determines the mapping of the file namespace to URLs. For example, the URL `http://dir/foo.htm` is mapped to the file `/usr/apache-docs/dir/foo.htm` by the apache server.

The reason for separating the file namespace from the server software and the URL mapping is to give us portability and generality. File names might more usefully be arranged by file owner, or by subject. We would like to be able to apply any appropriate interpreter to a file of a given type rather than being tied to one server. What is a server which uses the Netscape HTTP server to do with HTTP files stored in `/usr/apache-docs` and `/usr/cern-docs`? In order to provide generality, we must make the interpretation types of files both explicit and general.

3.2.1 Channel Metadata

The definition of a channel metadata standard for specifying the interpretation of distributed files has been addressed by a number of projects based in the Web industry. These XML-based schemas have been developed for desktop push and server content subscription. An adequate analysis and comparison of

these proposed standards is outside the scope of this paper. In lieu of a deeper analysis, we list a few notable efforts.

Resource Definition Framework — The Resource Definition Framework (RDF) is an XML-based metadata framework being developed by the W3C to support the exchange of machine-understandable information on the Web [20].

Internet Content Exchange — Internet Content Exchange (ICE) was initiated by Vignette, and is supported by an industry consortium including Adobe Systems, Microsoft, and Sun Microsystems, with the intent of creating a standard for delivery of syndicated Web source content to servers by subscription [21].

WebDAV — The Web Development and Versioning System was developed to give a standard network interface to Web source files stored by a server. One important feature of WebDAV is that source files are named by the same URLs used to request them, obviating the need for any knowledge of the server's file system structure [22].

3.2.2 Digital Object Representation

While the channel metadata standards discussed in Section 3.2.1 specify many attributes of the interpretation of distributed files, they are only be as expressive as the type system used to describe files, which is typically the MIME type system.

The MIME type system is widely used as metadata for SMTP and HTTP. MIME is an adequate standard for a limited set of types, but suffers from being limited to base types, and the only globally known types are those approved by the IETF. MIME does not support notions of type constructor or object hierarchy. Every new MIME object type must define a new base type, regardless of whether it is a straightforward combination of existing types or is even subsumed in an existing type except for some semantic difference.

Java object serialization is one appealing mechanism for transferring objects across the network in a format that is independent of both architecture and operating system. Because it defines a secure and portable mechanism for representing data objects and executable code, Java is certainly expressive enough to represent any digital object we might need to replicate as part of our channel content model. This approach was tried by the Marimba Castinet system but may have been hampered by the complexity of Java programming as a paradigm for content authoring. For executable content and structured objects, Java is a highly appropriate network standard, as are some other interpreted languages such as Tcl [23].

XML [24, 25] is an approach to layering structure on Internet content. XML allows us to describe relationships between objects through metadata. This approach is less appropriate for very fine-grained object structure, but it is declarative and easier to author than Java.

4. I2-DSI Project Milestones and Goals

The I2-DSI is a joint project of the University of Tennessee's Innovative Computing Laboratory, the University of North Carolina's School of Information and Library Science and the University Corporation for Advanced Internet Development It receives funding from the North Carolina Networking Initiative, the Digital Library Federation and seven industrial sponsors: Cisco Systems, Ellemtel, IBM, Novell, StorageTek, Starburst Communications, and Sun Microsystems. The project is in the early stages of deployment, with 6 servers installed and 2-4 more expected by 2Q99 equipped with terabyte storage systems. IBM and StorageTek are have made major commitments of server hardware and software and Starburst is donating reliable multicast software which will be used for experimentation in the use of multicast to synchronize distributed servers. Ellemtel is lending its engineering efforts to the development of I2-DSI server software. More details can be found at `http://dsi.internet2.edu`.

Interest in the I2-DSI project is high in the international networking community: active cooperation with CANARIE and CA*net II in Canada, initial discussions with European and Japanese corporations and with the TransEuropean Research Networking Association (TERENA). In working with these partners we are immediately faced with the issues of heterogeneity which are the subject of this paper. While it will be possible to place a few I2-DSI core servers within foreign networks (in cooperation with our equipment sponsors and foreign academic partners) we expect that there will quickly be an interest in exchanging channel content between independently operated storage infrastructures. For example, Nippon Telephone and Telegraph (NTT) of Japan has defined a multimedia network architecture and are interested in interoperating with I2-DSI. Thus, international cooperation is likely to be a major driving force in the development of portable channel content standards and interoperable interfaces.

5. Conclusions

In this paper we have elaborated a unified view of replication and resolution in Internet information systems which encompasses caching and replicated servers. In the context of this analysis we have

analyzed how the requirement for portable content representation of Internet content is addressed within the cache hierarchies, and we have shown that it must be addressed on systems of replicated servers such as I2-DSI. Unification of caching and replication is an important topic currently receiving attention through efforts to form an IETF Web Replication (WREC) working group. It is our belief that by revisiting the engineering foundations of both caching and replication we can develop a single paradigm for discussion and interoperation.

Works Cited

1. Beck, M. and T. Moore, *The Internet2 Distributed Storage Infrastructure Project: An Architecture for Internet Content Channels.* Computer Networking and ISDN Systems, 1998. **30**(22-23): p. 2141-2148.
2. Bowman, C.M., *et al. The Harvest Information Discovery and Access System*. in *The Second International WWW Conference*. 1994. Chicago, IL.
3. Wessels, D. and K. Claffy, *Internet Cache Protocol (ICP) version 2*, 1997.
4. Moore, K., *Transparent Caching Considered Harmful*, 1999.
5. Inktomi, *Sreaming Media Caching Brief.*
6. Cormack, A., *Caching on Janet*,. 1996, University of Wales: Cardiff, Wales.
7. Ellerman, C., *Channel Definition Format (CDF)*, 1997, Microsoft.
8. Hoff, A.v., *The HTTP Distribution and Replication Protocol*, 1997.
9. Transarc, *Managing Diversity in Wide-Area File Systems*, . 1998.
10. Wahl, M., T. Howes, and S. Kille, *Lightweight Directory Access Protocol (v3)*, 1997.
11. Tridgell, A. and P. Mackerras, *The rsync algorithm*, 1998, Australian National University: Canberra, Australia.
12. Weiss, D., *An Efficient, Scalable Replication Mechanism for the I2-DSI Project*, in *School of Information and Library Science*. 1999 (May, expected), University of North Carolina: Chapel Hill.
13. Mankin, A., *IRTF Reliable Multicast Group*.
14. Donnelley, J. *WWW Media Distribution via Hopwise Reliable Multicast.* in *3rd WWW Conference*, 1995. Darmstadt, Germany.
15. Delgadillo, K., *Cisco DistributedDirector*.
16. Gage, C., *Olympic-scale TCP/IP Load Balancing and Availability*, . 1998, IBM.
17. Swany, M., *sonarDNS*, . 1998.
18. Moat, R., *URN Syntax*, 1997.
19. CRNI, *The Handle System*.
20. Miller, J., T. Berners-Lee, and R.R. Swick, *Resource Description Framework (RDF)*, W3C.
21. Webber, N., *et al.*, *The Information and Content Exchange (ICE) Protocol*, 1998, W3C.
22. Goland, Y.Y., *et al.*, *HTTP Extensions for Distributed Authoring -- WEBDAV*, 1998, IETF.
23. Ousterhout, J.K., *Tcl and the Tk Toolkit*. 1994, Reading, MA: Addison-Wesley.
24. Bray, T., J. Paoli, and C.M. Sperberg-McQueen, *Extensible Markup Language (XML) 1.0*, 1998, W3C.
25. Bray, T., *Namespaces in XML*, .