WebOS: Operating System Services for Wide Area Applications*

Amin Vahdat† Eshwar Belani† Paul Eastham† Chad Yoshikawa†
Thomas Anderson‡ David Culler† Michael Dahlin§

Abstract

In this paper, we argue for the power of providing a common set of OS services to wide area applications, including mechanisms for resource discovery, persistent storage, remote process execution, resource management, authentication, and security. On a single machine, application developers can rely on the local operating system to provide these abstractions. In the wide area, however, application developers are forced to build these abstractions themselves or to do without. This ad-hoc approach often results in individual programmers implementing non-optimal solutions, wasting both programmer effort and system resources. To address these problems, WebOS provides basic operating systems services needed to build applications that are geographically distributed, highly available, incrementally scalable, and dynamically reconfiguring. Experience with a number of applications developed under WebOS indicates that it simplifies system development and improves resource utilization. In particular, we use WebOS to implement Rent-A-Server to provide dynamic replication of overloaded services across the wide area in response to client demands.

1 Introduction

While the World Wide Web has made geographically distributed read-only data easy to use, geographically distributed computing resources remain difficult to access. As a result, wide area applications that require access to remote CPU cycles, memory, or disk must be programmed in an ad hoc and application-specific manner. For example, many popular services, such as Digital's Alta Vista [Dig 1995] or Netscape's download page [Net 1994], are geographically replicated to improve bandwidth, reduce latency, and improve availability—no single connection onto the Internet can support tens of millions of users. Today, such replication is manually managed on both the server and the client side—users are forced to poll several essentially equivalent services and system managers must manually distribute updates to replicas. This situation will only get worse; it is currently predicted that the number of Internet users will increase by an order of magnitude to over 100 million in less than 5 years [Rutkowski 1995].

To address these problems, we have built WebOS, a framework for supporting geographically distributed, highly available, incrementally scalable, and dynamically reconfiguring applications. WebOS includes mechanisms for resource discovery, persistent storage, remote process execution, resource management, authentication and security. We use WebOS to demonstrate the synergy of these services in simplifying the development of wide area distributed applications and in providing more efficient global resource utilization. The WebOS framework enables a new paradigm for Internet services. Instead of being fixed to a single location, services can dynamically push parts of their responsibilities out onto Internet computing resources, and even all the way to the client.

Dynamically reconfiguring and geographically mobile services provide a number of advantages, including: (i) better end-to-end availability (service-specific extensions running in the client mask Internet or server failures), (ii) better cost-performance (by dynamically moving information closer to clients, network latency, congestion, and cost can all be reduced while maintaining server control), and (iii) better burst behavior (by dynamically recruiting resources to handle spikes in demand). For example, many Internet news services were overwhelmed on the night of the last U.S. presi-

*This work was supported in part by the Defense Advanced Research Projects Agency (N00600-93-C-2481, F30602-95-C-0014), the National Science Foundation (CDA 9401156), Sun Microsystems, California MICRO, Novell, Hewlett Packard, Intel, Microsoft, and Mitsubishi. Anderson was also supported by a National Science Foundation Presidential Faculty Fellowship. For more information, please see `http://now.cs.berkeley.edu/WebOS`.

†Computer Science Division, University of California, Berkeley

‡Department of Computer Science and Engineering, University of Washington, Seattle

§Computer Science Department, University of Texas, Austin

dential election; our framework would enable those services to handle the demand through dynamic replication. One argument against this approach is that service providers will not trust geographically distributed compute servers. However, many popular Internet services already are statically configured to run at third party sites to pro-rate the cost of a fault tolerant, high bandwidth Internet connection over multiple service providers [Brewer 1997]; we are merely arguing that with the right support, these arrangements can be made dynamically and managed automatically.

In addition to demonstrating the synergy of a common framework for wide area distributed applications, we make a number of specific contributions. First, we demonstrate an extensible mechanism for running service-specific functionality on client machines and show that this allows for more flexible implementation of name resolution, load balancing, and fault tolerance. Second, we provide a file system abstraction that combines persistent storage with efficient wide-area communication patterns; we demonstrate that this simplifies the implementation of a number of wide area applications, including Internet chat and a remote compute server. Finally, we motivate and describe our implementation of Rent-A-Server, an application that allows for transparent, automatic, and dynamic replication of HTTP service across the wide area in response to client load. Rent-A-Server also demonstrates the power of exporting common operating system abstractions to wide area applications; WebOS services simplified both the design and implementation of Rent-A-Server.

The rest of this paper discusses these issues in more detail. We present an overview of WebOS system components in Section 2, before delving into more detail in Sections 3-5, which describe: (i) Smart Clients and Smart Proxies for fault tolerant, load balanced access to Web services, (ii) WebFS, a global cache coherent file system, (iii) a process control model supporting secure program execution and mechanisms for resource allocation, and (iv) authentication for secure access to global Web resources. Section 7 demonstrates how this framework simplifies the implementation of four sample wide area applications. Section 8 describes in detail the design, implementation, and performance of one application built on WebOS, Rent-A-Server. Section 9 presents related work, leading to our conclusions in Section 10.

2 WebOS Overview

In this section, we provide a brief overview of the major WebOS components; together, they provide the wide area analogue to local area operating system services, simplifying the use of geographically remote resources. Each of these components is operational in our current prototype.

- *Resource Discovery:* Many wide area services are geographically distributed. To provide the best overall system performance, a client application must be able to dynamically locate the server able to deliver the highest quality of service. In WebOS, resource discovery includes mapping a service name to multiple servers, an algorithm for balancing load among available servers, and maintaining enough state to perform fail-over if a server becomes unavailable. These operations are performed through Smart Clients and Smart Proxies, which flexibly extend service-specific functionality to the client machine.

- *Wide Area File System:* To support replication and wide-scale sharing, WebOS provides a cache coherent wide area file system. WebOS extends to wide area applications running in a secure HTTP name space the same interface, caching, and performance of existing distributed file systems [Walsh et al. 1985, Nelson et al. 1988, Howard et al. 1988, Kistler & Satyanarayanan 1992, Terry et al. 1995, Anderson et al. 1995]. In addition, we demonstrate the benefit of integrating the file system with application-controlled efficient wide area communication.

- *Security and Authentication:* To support applications operating across organizational boundaries, WebOS defines a model of trust providing both security guarantees and an interface for authenticating the identity of principals. A key enabling feature is fine-grained control of capabilities provided to remote processes executing on behalf of principals.

- *Process Control:* In WebOS, executing a process on a remote node should be as simple as the corresponding local operation. The underlying system is responsible for authenticating the identity of the requester and determining if the proper access rights are held. Precautions must be taken to ensure that the process does not violate local system integrity and that it does not consume more resources than allocated to it by local system administrators.

As an explicit design choice, we leverage as much functionality as possible from existing low level services. For example, for compatibility with existing applications, we adopt IP addresses and URL's for the global namespace, TCP to provide reliable communication, and SSL [Freier et al. 1996] for link level security.

3 Resource Discovery

In this section, we discuss how WebOS clients locate representatives of geographically distributed and dynamically reconfiguring services, while providing load balancing and end-to-end high availability.

Our approach consists of a number of components. First, a service name must be mapped onto the replicated service representatives. Next, a load balancing decision must be made to determine which server is able to deliver the best performance; this evaluation is dynamic and non-binding to cope with potentially bursty client access patterns. Finally, enough state (e.g. request content) is maintained to perform fail over if a service provider becomes unavailable. This section describes limitations associated with current approaches to resource discovery and shows how WebOS addresses these limitations through the use of Smart Clients and Smart Proxies. Our discussion focuses on resource discovery in the context of HTTP service accessed through URL's. However, our techniques extend to other domains in a straightforward manner.

3.1 Current Approaches

To address increasing demand, some popular services such as the Alta Vista search engine [Dig 1995] or the Netscape download page [Net 1994] are geographically distributed by being replicated manually by the service provider. Load balancing across the wide area is achieved by instructing users to access a particular “mirror site” based on their location. To distribute load across servers, techniques such as HTTP redirect [Berners-Lee 1995] or DNS Aliasing [Brisco 1995, Katz et al. 1994] can be used to send user requests to individual machines. With HTTP redirect, a front end machine redirects the client to resend the request to an available worker machine. This approach has the disadvantage of either adding a round trip message latency to each request or of binding the client to a single server for the duration of a session. Further, the front-end machine serving redirects is both a single point of failure and a central bottleneck for very popular services.

DNS Aliasing allows the Domain Name Service to map a single hostname (URL) to multiple IP addresses in a round robin fashion. Thus, DNS aliasing does not suffer from the added latency and central bottlenecks associated with HTTP redirect. However, currently load balancing with DNS aliasing is restricted to round-robin, making it difficult to use service-specific knowledge such as load information. Further, because clients cache hostname to IP address mappings, a single server can become overloaded with multiple requests while other servers remain relatively idle [Dias et al. 1996].

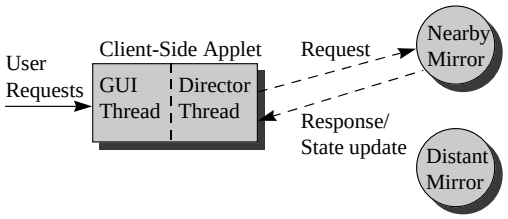


Figure 1: Two cooperating threads make up the Smart Client architecture. The GUI thread presents the service interface and passes user requests to the Director Thread. The Director is responsible for picking a service provider likely to provide best service to the user. The decision is made in a service-specific manner. In this case, the nearest mirror site is chosen.

3.2 Smart Clients

In WebOS, we address the shortcomings of existing solutions for resource discovery through Smart Clients [Yoshikawa et al. 1997]. Smart Clients enable extensions of server functionality to be dynamically loaded onto the client machine. Java's [Gosling & McGilton 1995] portability and availability in all major Internet browsers allow us to distribute these extensions as Java applets. We believe that performing naming, load balancing, and fail-over from the perspective of the client has a number of fundamental advantages over server-side implementations. We demonstrate these advantages by using our Smart Clients prototype to implement scalable access to services such as FTP, chat, and the Rent-A-Server application described in Section 8.

The Smart Client architecture is summarized in Figure 1. A typical applet's code is composed of two cooperating threads: a customizable graphical interface thread implementing the user's view of the service and a director thread responsible for performing load balancing among service representatives and maintaining the necessary state to transparently mask individual failures. Both the interface and director threads are extensible in a service-specific manner.

One outstanding issue with this architecture is the choice of load balancing algorithm. For example, a front end machine serving HTTP redirects to clients has the advantage of current knowledge of load on each worker process. In the case of Smart Clients, it is impractical to keep all clients abreast of changes in load of all servers. Given the high variability of load in the

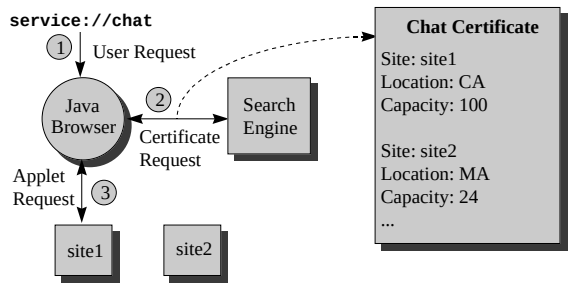


Figure 2: Bootstrapping applet retrieval in Smart Clients and Smart Proxies. A new service name space is introduced. In step 1, a name is translated into a certificate request from highly-available search engines in step 2. The certificate contains hints as to initial service group membership. The Smart Client applet is retrieved from one of these sites in step 3. Both the certificate and applet are cached to disk to avoid future bootstrapping.

context of the Internet, clients using out of date information may make strictly worse choices than clients making random decisions [Mitzenmacher 1996]. Fortunately, Smart Clients can use more static state information to influence the load balancing decision, such as available servers, server capacity, server network connectivity, server location, and client location.

While the specific load balancing algorithm is service-specific, we implement the following algorithm by default. Service state is piggy-backed with some percentage of server responses (i.e. as part of the HTTP header). The client then chooses a server based on distance from the client, biased by server load. The influence of load information is decayed as the information becomes stale, with the fallback to random load balancing in the case where load information is stale and all other considerations are equal. Thus, clients actively interacting with a service can use current information to make load balancing decisions, while inactive clients must initially discount their outdated notion of load.

3.3 Bootstrapping Applet Retrieval

While the Smart Client architecture provides a portable mechanism for fault tolerant and load balanced access to Web services, bootstrapping Smart Client startup remains to be described. Naively, services would be named through URL's, with the applet downloaded each time the service is to be accessed. This would imply a central bottleneck, a single point of failure, and effectively doubling latency for small requests.

Figure 2 summarizes our approach to addressing these issues. A *meta-applet* runs in a Java enabled browser and is responsible for bootstrapping accessing to Web services. A new service namespace is introduced, allowing users to make requests of the service URL's [Veizades et al. 1997]. The meta-applet translates these names into requests to a well-known and highly available Internet name service to fetch a *service certificate*—the list of servers capable of providing the service, along with individual server characteristics. Currently, we use Alta Vista to perform this operation, however we plan to switch to a more lightweight technique, such as DNS [Mockapetris & Dunlap 1988]. The service's Smart Client applet is downloaded from one of these sites and operation proceeds normally as described in Section 3.2. The service certificate, the Smart Client applet, and any service state are cached to disk (with timeouts) by the meta-applet to avoid bootstrapping on subsequent accesses.

3.4 Smart Proxies

While using Smart Clients for load balancing, naming, and fault tolerance presents a number of compelling advantages, it also faces a number of limitations. First, deployment can be difficult since clients must choose applets that display potentially dangerous behavior. For example, the applet must be able to make network connections to arbitrary network hosts, an operation that users must explicitly allow in current browser security models [Gong 1997, Wallach et al. 1997]. Another limitation is that unless clients are frequently interacting with a service, their notion of load is likely to become quickly out-dated. While using static service information as described above to make load balancing decisions is still useful, clearly using updated load information in conjunction with more static state is preferable.

One approach to addressing both these limitations is through the deployment of *Smart Proxies*. Proxies have the advantage of aggregating the behavior of a large number of clients. Proxies simplify deployment since service access applets can be downloaded the first time any proxy client accesses a service; subsequent accesses by a different clients can used cached copies of the applet. Further, security concerns with applets can be addressed once by system administrators rather than forcing individual users to determine the security risks associated with applet behavior. Applets run through Smart Proxies also have the advantage of improved load balancing information. Since a proxy aggregates the access patterns of potentially thousands of clients, the probability of possessing current information for access to popular services increases dramatically.

Of course, Smart Clients possess their own set of advantages relative to a proxy-side approach. For exam-

ple, a client-side approach has the advantage of better knowledge of client state. Information such as the state of the client's Internet connection (modem, wireless, T1, etc.) or the size of the client display (PDA, laptop, workstation monitor) can be used to negotiate for different versions of the same object (postscript, PDF, text) based on available viewers, bandwidth, and screen size [Fox et al. 1997]. To investigate the relative advantages of Smart Clients and Proxies, we are in the process of modifying the Apache web server to include Smart Proxy functionality.

4 Persistent Storage

Today, many distributed applications share state and transfer control using the network communication abstraction. Using the analogy that it is often simpler to program parallel applications using shared memory as opposed to message passing, the implementation of many applications can be simplified by the use of a global cache coherent file system for communication and synchronization. As empirical evidence, the implementation of an Internet chat (Section 7.1) was simplified by encapsulating conversations in secure globally accessible files. Further, many wide-area applications require must perform caching to achieve reasonable performance. Currently however, any desired consistency guarantees must be layered on top of protocols such as HTTP or FTP. Rent-A-Server (Section 8) demonstrates the utility and performance available from a cache-consistent global file system. In this section we describe the implementation of our global file system.

WebFS is our prototype of a URL-based cache coherent file system. WebFS has been in day to day use for approximately six months by twenty users at the authors' site. It is publically available for download and has been successfully installed by a number of users unaffiliated with the authors. WebFS allows UNIX programs to take URL's (and URL's in a pathname syntax) in place of conventional file names (e.g., `ls /http/www.usenix.org/events/`). We chose to use URL's as the global namespace because of its wide deployment and our desire to provide backward compatibility with existing distributed applications. Since it is preferable to export a namespace with location independent names, we are currently investigating combining Smart Proxies with WebFS to provide URN [Sollins & Masinter 1994] support for WebFS file names.

A fundamental difference between WebFS and existing Internet naming and caching proposals is that WebFS is designed to be used by programs, not just by individuals accessing widely-shared, infrequently updated data. Web users have demonstrated tolerance for out-of-date data with manual revalidation. By contrast, we believe distributed applications require complete and well-defined file system semantics. Since the choice of consistency semantics implies tradeoffs in performance, consistency, and availability, WebFS provides application-controllable and dynamically adaptable cache consistency policies. For example, invalidation driven optimistic consistency could be used for files shared between two geographically distributed sites implementing a service [Kistler & Satyanarayanan 1992, Terry et al. 1995], while weaker consistency could be used for widely-shared files at client machines. As another example, to support service migration, WebFS may initially use large block sizes or prefetching to reduce cold-start misses and then convert to smaller blocks for minimizing false sharing. Finally, a list of user-extensible properties is associated with each WebFS file, extending basic properties such as owner and permissions with cache consistency policy and encryption policy. These properties are set and accessed through the UNIX `ioctl` system call.

Currently, WebFS implements the last writer wins [Howard et al. 1988] cache consistency protocol to support traditional file access as well as an IP multicast-based [Deering 1991] update/invalidate protocol for widely-shared, frequently updated data files. Once IP multicast becomes widely deployed, its use will increase the efficiency of popular “Internet push” applications [Poi 1996]. We believe that providing IP multicast support at the file system interface will simplify the development of these applications. To demonstrate this point, we have implemented a stock ticker application that regularly distributes (through multicast file writes) updated stock prices to interested clients performing blocking read operations. In addition to last-writer wins and IP multicast updates, we are in the process of extending WebFS to support optimistic re-integration [Kistler & Satyanarayanan 1992] to deal with the disconnected operation endemic to today's Internet.

WebFS is implemented as a dynamically loadable Solaris file system extension interfacing at the vnode layer [Kleiman 1986]. The vnode layer makes upcalls to a user level WebFS daemon for file accesses not cached in virtual memory. The WebFS daemon uses HTTP for access to standard Web sites. If the remote site is also running WebFS, then authenticated read/write access is enabled through our own custom extensions to HTTP. The performance of remote file access is highly dependent upon the network connection to the remote site. The cost of the network connection and subsequent transfer dominates the added overhead of making an upcall to the user-level. Once transferred through the network, file pages are cached in the kernel

file cache. Thus, once a file has been transferred from a remote site, the performance of cached access through WebFS vs. cached access through NFS [Walsh et al. 1985] is virtually identical.

5 Process Control

To simplify development of wide area applications, WebOS makes execution of processes on remote nodes as simple as forking a process on the local processor. As with the local case, the WebOS process control model addresses issues with safety and fairness. On local machines, safety is provided by execution in a separate address space, while fair allocation of resources is accomplished through local operating system scheduling mechanisms.

A *resource manager* on each WebOS machine is responsible for job requests from remote sites. Before executing any job, the resource manager authenticates the remote principal's identity and determines if the proper access rights are held. To maintain local system integrity and to ensure that running processes do not interfere with one another, the resource manager creates a *virtual machine* for process execution.

We use Janus [Goldberg et al. 1996] to create such a virtual machine. Processes in the virtual machine execute with limited privileges, preventing them from interfering with the operation of processes in other virtual machines. Janus uses the Solaris `/proc` file system to intercept the subset of system calls that could potentially violate system integrity, forcing failure if a dangerous operation is attempted. A Janus configuration script determines access rights to the local file system, network, and devices. These configuration scripts are set by the local system administrator on a per-principal basis.

WebOS also uses the virtual machine abstraction as the basis for local resource allocation. On startup, a process's runtime priority is set using the System V `priocntl` system call, and `setrlimit` is used to set the maximum amount of memory and maximum CPU usage. In the future, we hope to integrate more robust policies allowing fine-grained control over allocation, allowing WebOS to provide quality of service guarantees. For example, techniques such as reverse lotteries [Waldspurger & Weihl 1994] might be used to more flexibly allocate physical memory pages.

6 Security and Authentication

Applications operating across the wide area are susceptible to a variety of potential attacks by sophisticated adversaries. For users to trust their sensitive computation to WebOS, a well-defined and easily validated model of trust must be defined and implemented. The goal of our implementation is to make issues of security as transparent as possible for common operations while still allowing for arbitrary levels of security for programs that require it. CRISIS, the security system of WebOS, is described in another publication [Belani et al. 1998]; we present an overview of CRISIS here.

The CRISIS architecture is based on the theory presented in [Lampson et al. 1991, Wobber et al. 1993] and thus assumes the presence of *principals*, sources for requests such as machines or users, *objects*, global resources such as files, processors, printers, and memory, and *reference monitors*, processes that determine whether or not to grant a given request. In this section, we describe the CRISIS mechanisms for authenticating principals, making requests, and determining access rights.

6.1 Validating and Revoking Statements

All statements in CRISIS, including statements of identity, statements of privilege, and transfer of privilege, are encoded in *certificates*. CRISIS certificates are signed by the principal making the statement and then counter-signed by a principal of the signer's choosing. Each signature uses a separate timeout: the principal's signature is issued with a long timeout, while the counter-signature is issued with a short timeout. The counter-signer (i) checks if the statement has been revoked and (ii) refreshes its *endorsement* (by applying a new counter-signature with a new timeout to an expired certificate) of certificates, indicating that the rights are are still valid. CRISIS is based on public key cryptography; thus a certificate's author need not be aware of the certificate's destination when it is created. Any principal with access to the certificate can determine the statement's author. Our certificates use the X.509 [Con 1989] standard format.

CRISIS employs two basic types of certificates. *Identity Certificates* associate a public key with a principal for a certain period of time. An identity certificate can also specify a number of a principal's properties, such as name or organization. *Transfer Certificates* transfer a subset of a principal's privileges to another principal. A principal P_1 can use a transfer certificate to transfer to P_2 access rights to any objects it owns. These transfers are expressed as a list of capabilities, resulting in arbitrary length certificates. Chains of transfer certificates are presented to reference monitors as proof of access rights.

Identity certificates must be signed by an authority trusted by both endpoints of a communication channel. This trusted third party, called the Certification Au-

thority (CA), maps public keys to principals and maintains a Certificate Revocation List enumerating all public keys that have changed or that have been knowingly compromised. In CRISIS, CA's sign all identity certificates with a long timeout (usually weeks) and identify a locally trusted on-line agent (OLA) responsible for counter-signing the identity certificate with a relatively short timeout (usually hours). Redundancy employed in this fashion offers a number of advantages: (i) to successfully steal keys, either both the OLA and CA must be subverted or the CA must be subverted undetected, (ii) the CA is usually left off-line since certificates are signed with long timeouts, increasing system security since an off-line entity is more difficult to attack, (iii) a malicious CA is unable to revoke a user's key, issue a new identity certificate, and masquerade as the user without colluding with the OLA [Crispo & Lomas 1996], and (iv) system performance is improved because certificates can be cached for the timeout of the counter-signature, removing the need for synchronous three-way communication in the common case.

We generalize the OLA to make revocation a first class operation in CRISIS. All certificates are revocable modulo a timeout. To revoke a particular privilege, the OLA which endorses the certificate must be informed that the certificate should no longer be endorsed. Once the timeout period for the endorsed certificate expires, the rights described by the certificate are effectively revoked because the OLA will refuse re-endorsement for that certificate. Revocation is used not only for exceptional events such as stolen keys, but also applies to common operations such as revoking the rights of a remote job upon its completion or revoking the rights of a login session upon user logout.

One outstanding issue with the above discussion is the real-world concern that the Web is made up of many mutually distrustful administrative domain. To address this issue, we assume that representative CA's for each administrative domain are arranged hierarchically, with individual CA's determining which parents, siblings, or children are trusted (and to what extent). The manner in which the hierarchy is traversed is based on the theory presented in [Birrell et al. 1986]. In this model, trust extends only as far as needed. To establish secure communication within a domain, only the local CA for the domain need be contacted. Non-root CA's in the hierarchy are used only to facilitate communication between principals in separate domains, providing for a *path of trust* through the least common ancestor CA of the communicating principal's two domains.

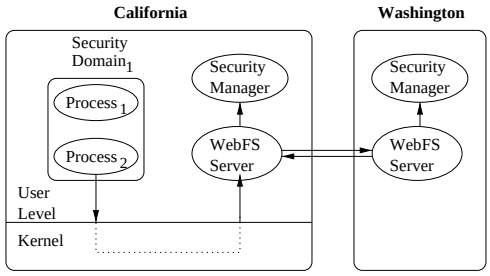


Figure 3: This figure describes the interaction of resource providers, security managers, and security domains for the example of file access.

6.2 Processes and Roles

Given the abilities to authenticate principals, CRISIS also requires a mechanism for associating privileges with running processes. Each CRISIS node runs a security manager responsible for mediating access to all local resources and for mapping credentials to *security domains*. In CRISIS, all programs execute in the context of a security domain. For example, a login session creates a new security domain possessing the privileges of the principal who successfully requested login. A security domain, at minimum, is associated with a transfer certificate from a principal to the local node allowing the node to act on the principal's behalf for some subset of the principal's privileges.

Figure 3 summarizes the interaction between resource providers, security domains, and security managers for file access. In this example, a process running in California requests a file physically located in Washington. The request is forwarded to a user-level WebFS server through the kernel. The server contacts the local security manager to determine the process's security domain and its associated credentials. The California WebFS server transmits the file request along with the credentials of the calling process to the WebFS server in Washington. The Washington WebFS server in turn contacts its local security monitor to determine authorization (described below). If access is granted, the requested file is transmitted back to California.

In the wide area, it is vital for principals to restrict the rights they cede to their jobs. For example, when logging into a machine, a principal implicitly authorizes the machine and the local OS to speak for the principal for the duration of the login session. A principal (user) creates a new role by generating an identity certificate containing a new public/private key pair and a transfer certificate that describes a subset of the principal's rights that are transferred to that role; an OLA chosen by the principal is responsible for endorsing the cer-

tificates. Thus, in creating new roles, principals act as their own certification authority [Rivest & Lampson]. The principal stores the role identity certificate and role transfer certificate in a *purse* of certificates that contains all roles associated with the principal.

6.3 Authorization

Once a request has been securely transmitted across the wide area, and properly authenticated, the remaining task is *authorization*, determining whether the principal making the request should be granted access. CRISIS employs Access Control Lists (ACLs) to describe the principals and groups privileged to access particular resources. File ACLs contain lists of principal's authorized for read, write, or execute access to a particular file. Process execution ACLs are a simple list describing all principals permitted to run jobs on a given node.

To determine whether a request for a particular operation should be authorized, a reference monitor first verifies that all certificates are unexpired and signed by a public key with a current endorsement from a trusted CA and OLA. In doing so, the reference monitor checks for a path of trust between its home domain and the domains of all signing principals. The reference monitor then reduces all certificates to the identity of single principals. For transfer certificates, this is accomplished by working back through a chain of transfers to the original granting principal. Finally, the reference monitor checks the reduced list of principals against the contents of the object's ACL, granting authorization if a match is found.

7 WebOS Applications

This section provides an overview of four applications designed using the WebOS framework. The first two applications have been completed, while the last two are under development. In the next section, we describe in detail the design and performance of a fifth application, Rent-A-Server.

7.1 Internet Chat

Internet chat allows for individuals to enter and leave chat rooms to converse with others co-located in the same logical room. In our implementation, chat rooms are implemented as WebFS files accessed by Smart Clients. The file system interface is well-matched to chat semantics in a number of ways: (i) A simple file append abstracts the required network communication necessary to send messages, (ii) the chat file provides a persistent log of chat activity, and (iii) access control lists allow for private and secure (through WebFS encryption) chat rooms. For scalability, we allow multiple WebFS servers to handle client requests for a single file (room). Each WebFS server accumulates updates, and periodically propagates the updates to other servers in the WebFS group, who in turn transmit the updates to local clients. Smart Clients choose the least loaded WebFS server for load balancing and connect to alternative servers on host failure or network partition for fault transparency.

To quantify the benefits available from the WebOS framework, we implemented two versions of chat with identical semantics, both with and without WebOS. The initial implementation consisted of 1200 lines of Java code in the client and 4200 lines of C++ code in the server. By using WebFS to handle message transmission, failure detection, and storage, the size of the chat client code was reduced to 850 lines, while the WebFS interface entirely replaced the 4200 lines of chat server code. The main reason for this savings in complexity was the replacement of separate code for managing communication and persistent storage of chat room contents with a single globally accessible and consistent file. As an added benefit, this common WebFS interface is similarly available for other distributed applications. For example, we are currently implementing a shared distributed whiteboard application using this interface.

7.2 Remote Compute Engine

Sites with unique computing resources, such as supercomputer centers, often wish to make their resources available over the Internet. Using WebOS, we allow remote programs to be invoked in the same way as local programs and can allow access to the same files as local programs. WebOS functionality is used to address a number of issues associated with such access: the identity of requesting agents is authenticated, programs are provided secure access to private files on both local and remote systems, and programs run in a restricted virtual machine isolated from other programs to protect the local system from malicious users. At our site, WebOS provides compute access to a research cluster of 100 machines. Resource allocation within the virtual machine allows external users to take advantage of the aggregate computing resources, while ensuring system developers have the requisite priority.

7.3 Wide Area Cooperative Cache

We are using WebOS to build a geographically distributed Web cooperative cache [Dahlin et al. 1994] to both validate our design and to provide an immediate benefit to the Internet by doing more intelligent caching

of Web content. Existing proposals for hierarchical caching of the Web suffer from an inability to dramatically grow the cache size and processing power at each level of the hierarchy [Chankhunthod et al. 1996]. With cooperative caching among peer servers, the aggregate capacity grows dramatically with the distance from the client. Thus, while caches above the first level in existing hierarchical designs have very low hit rates and simply increase the latency to end clients, a cooperative cache is more likely to successfully retrieve a cached copy from a peer. We plan to explore tradeoffs associated with maintaining directories of peer cache contents [Anderson et al. 1995, Feeley et al. 1995], hints [Sarkar & Hartman 1996], or using simple IP multicasts or broadcasts.

WebOS simplifies the implementation of the cooperative cache in a number of ways. First, Smart Clients are used to determine the appropriate proxy cache to contact. WebFS is used to transport cache files among the proxies and to securely share any necessary (private) state among the proxies. Finally, the authentication model allows proxies to validate their identities both to one another and to the client.

7.4 Internet Weather

A number of sites are currently attempting to provide regular updates of congestion, latency, and partitions in the Internet [Mat 1996, Int 1997, Wolski 1997]. Such information is invaluable for services making placement and load balancing decisions. However, all current efforts take network measurements from a centralized site, making it difficult to measure network characteristics between two arbitrary sites. We are addressing this limitation by using the WebOS framework to generate more comprehensive snapshots of Internet conditions. In our implementation, a centralized server provides Smart Client applets for those wishing to view the current Internet weather. In exchange for the weather report, the user implicitly agrees to allow the applet to execute `traceroute` to a subset of server-determined sites and to transmit the result back to the server. Using these results from multiple sites, the service is able to construct fairly comprehensive snapshots of Internet weather.

8 Rent-A-Server

This section describes the design, implementation, and performance of Rent-A-Server, an application demonstrating the power of using a unified system interface to wide area resources and of moving a service out across the Internet.

8.1 Motivation

Rent-A-Server is a general model for graceful scaling across temporal and geographic spikes in client demand for a particular service. Our particular implementation focuses on Web service, and enables overloaded HTTP servers to shed load onto idle third-party servers called *surrogates* that use the WebOS framework to coherently cache data from the primary server. The surrogate is able to satisfy the same HTTP requests as the original server, including requests for both static and dynamically generated objects (e.g. data pages and CGI script results).

Rent-A-Server allows sites to deal with peak loads that are much higher than their average loads by "renting" hardware to deal with peaks. For example, the Internal Revenue Service site (`http://www.irs.ustreas.gov`) is overwhelmed by requests around April 15, but providing the computation power and network bandwidth necessary to handle peak levels of demand year round is a waste of resources. The benefits for Rent-A-Server can be summarized as follows:

- *Geographic Locality*: In addition to distributing load, Rent-A-Server improves performance by increasing locality. Rather than satisfying requests from a centralized site, a system can distribute its requests across geographically distributed servers, each of which satisfies nearby clients.

- *Dynamic Reconfiguration*: The location and number of sites representing a service can be determined dynamically in response to client access patterns. Rent-A-Servers can be spawned to locations "near" current spikes in client demand, and torn down once client activity subsides.

- *Transparent End-to-End Availability*: Once a service is replicated with Rent-A-Server, users can transparently access whichever replica is available, routing around both Internet and service failures.

- *Secure Coherent Data Access*: To address limitations associated with caching proxies which are unable to generate dynamic Web pages (e.g. results of cgi-bin programs) and often serve stale data, Rent-A-Server uses WebOS to provide authenticated, coherent global file access to data pages, CGI scripts, and internal server state needed by CGI scripts.

- *Safe Remote Execution*: Surrogate sites securely execute service programs and associated service scripts (such as CGI programs) without violating the surrogate's system integrity.

8.2 Current Approaches

Current efforts to distribute HTTP server load focus on either distributing load across a fixed set of machines maintained by the owner of the data or distributing data across (proxy) caches under client (not server) control. Many HTTP server implementations achieve scalability by replicating their data across a fixed set of servers at a single site and then using the Domain Name Service (DNS) to randomly distribute requests across the servers [Katz et al. 1994]. Unfortunately, this approach requires that each site purchase enough computing power and network bandwidth to satisfy peak demand.

"Mirror sites" are also used to improve locality and to distribute load, but this manual approach requires more effort to set up the mirrors and to maintain data consistency across the mirrors. Further, users must specify which mirror to use, which is both inconvenient and unlikely to yield a balanced load across sites. Finally, as with the approach of running multiple servers at one site, mirror sites are allocated statically. The system must always maintain enough mirrors to deal with its peak loads, and the location of mirrors cannot be shifted to address shifts in geographic hotspots.

Another approach to distributing load, caching proxies, is used to reduce server load and to improve network locality. To use a proxy, groups of clients send all of their requests to their proxy machine. The proxy machine attempts to satisfy the requests from its local cache, sending the requests to the remote server if the cache cannot supply the data. If proxies satisfy many requests to the server through their caches, both server load and network congestion are reduced.

However, proxies are conceptually agents of Web clients rather than of Web servers. Thus, in some instances they provide a poor match to the requirements of overloaded services. First, proxy servers cache only data pages. A proxy must send all requests for CGI scripts to the original server. Second, because servers regard proxies as ordinary clients, the proxy can supply stale data to its clients because of the limitations of HTTP cache consistency protocols. As an example of the importance of having server-controlled rather than client-controlled load distribution, some sites have recently asserted that proxy caches violate copyright laws by storing site content [Luotonen & Atlis 1994]. In effect, the proxies are reducing generated advertising revenues by hiding page access counts.

8.3 System Design

In this subsection, we demonstrate how WebOS services simplify the implementation of this application.

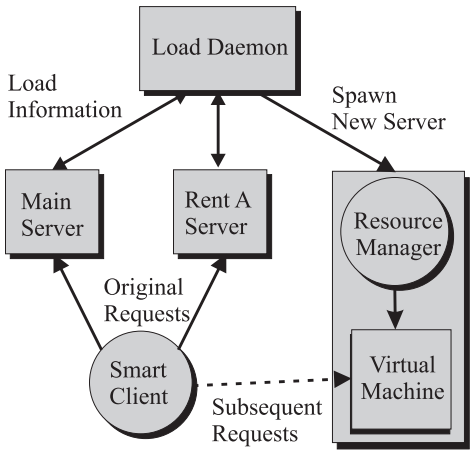


Figure 4: Rent-A-Server Architecture. HTTP servers periodically send load information to a load daemon. In response to an update, the load daemon transmits the state of all servers. In turn, the HTTP servers transmit this state information as part of the HTTP header to Smart Clients. The Smart Clients can use this information to determine which server to contact for its next request. When the load daemon notices that the service as a whole is becoming overloaded, it contacts the resource manager on an available surrogate to create another server replica. WebFS is used to securely transmit any executables or data files needed to start the server.

The architecture of the Rent-A-Server is described in Figure 4. Smart Clients access HTTP services. Periodically (currently every tenth response), servers piggyback service state information to Smart Clients in the HTTP reply header. This state information includes a list of all servers currently providing the service. The following information is included for each server: its geographic location, an estimate of its processing power, an estimate of current load, and a time period during which the server is guaranteed to be active. The last field is determined with short term leases that are periodically renewed if high demand persists. The short leases prevent clients with stale state information from trying to access inactive surrogates (or worse, surrogates acting on behalf of a different service).

Each Rent-A-Server maintains information about client geographic locations (location is sent by Smart Clients as part of the HTTP request) and its own load information in the form of requests per second and bytes transmitted per second. Each Rent-A-Server periodically transmits this state information to a centralized *load daemon*. For software engineering reasons, the load daemon is currently a separate process, however its functionality could be rolled into an elected member

of the server group. The load daemon is responsible for determining the need to spawn or to tear down Rent-A-Servers based on current load information and client access patterns. It also transmits server group state (e.g. membership and load information) to each member of the server group, which is in turn piggy-backed by the servers to Smart Clients as part of HTTP replies, as described above.

Once the load daemon determines the need to spawn an additional server, it first determines a location for the new Rent-A-Server. The new server should be located close to any hotspots in client access patterns to both conserve bandwidth and to minimize client latency (this policy has not yet been implemented). Once the target machine is selected, the load daemon establishes an SSL channel with the surrogate's resource manager. The load daemon then adds the surrogate to the necessary ACL's allowing it access to WebFS files containing the executables (e.g. HTTP server) or internal service state (e.g. CGI scripts or internal database). In addition, the load daemon provides a signed transfer certificate granting the surrogate the right to serve data on behalf of the service. This certificate is transmitted to Smart Clients on demand to prevent spoofing of the service by malicious sites; it is revoked when the surrogate is no longer needed to serve the pages.

When setup negotiation is completed, the surrogate site builds a Janus virtual machine to execute the necessary programs (in our case an arbitrary HTTP server) to establish a service identity at the surrogate. The virtual machine ensures that the surrogate's system integrity is not violated by a buggy executable or a malicious server. Both the service executable and any necessary service state are securely accessed and cached on demand through WebFS. The load daemon propagates the identity of the new surrogate to other members of the server group, which in turn transmit the identity and location of the new server to Smart Clients. Tear down of a surrogate is accomplished when client demand subsides and the load daemon decides not to renew leases with a surrogate. The load daemon removes the surrogate from the appropriate ACL's.

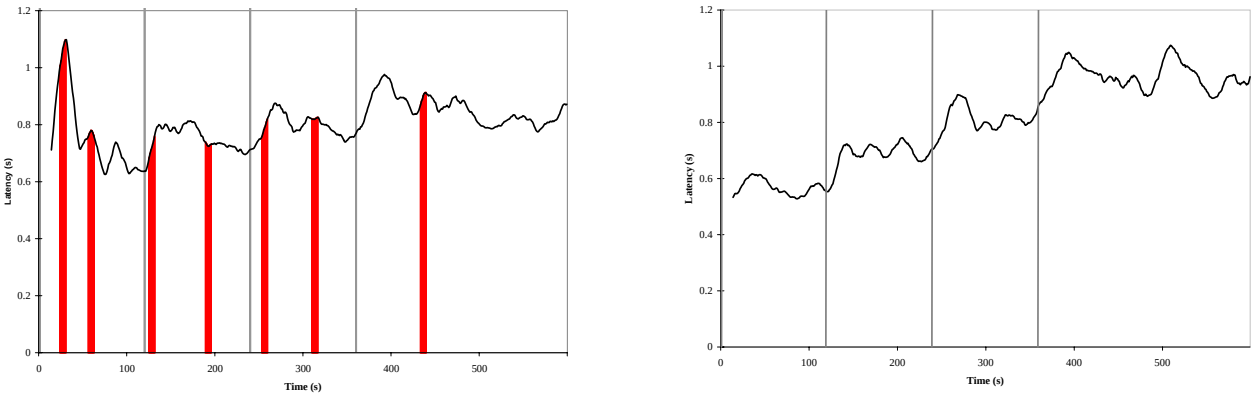
8.4 Performance

To demonstrate the power of dynamic resource recruitment available from our approach, we measure the performance of Rent-A-Server when placed under a heavy synthetic load. Our experiments are conducted on a cluster of Sun Ultra Servers interconnected by 10 Mbps switched Ethernet. Seven Ultra Servers are designated as surrogates available to provide HTTP service on behalf of an eighth machine designated as the primary. All eight server machines run WebOS, including the file system and the resource manager responsible for process control. Another 32 machines are used to generate gradually increasing load to the HTTP service. All machines are running Solaris 2.5.1, and Apache 1.2b6 is used for the HTTP server.

During the experiment, each client machine starts 15 Smart Client processes that continuously retrieve copies of the same 2.5 KB (a typical size today [Gribble & Brewer 1997]) HTML file. Smart Clients use random load balancing to pick from a changing list of available HTTP servers. To simulate increasing load, all 32 client machines do not begin making requests at the same time. Rather, clients start in four groups of eight machines each, with the start time of each group staggered by two minutes. Thus, all 32 clients are running after six minutes.

The results of our tests are summarized in Figure 5. The graphs plot average client-perceived latency in seconds as a function of elapsed time, also in seconds. Initially, only a single HTTP server is available; however, the load daemon spawns new servers onto available surrogates as service load increases. The thick dark columns correspond to execution of new HTTP servers, while the lighter thin lines correspond to the startup of a new group of eight client machines, with the first eight clients starting at time 0. For example, the Rent-A-Server graph shows that a third surrogate server was started at $t = 122$, shortly after the second group of eight clients start. The experiment reaches steady state at approximately $t = 500$ seconds after the last group of client machines start running at $t = 360$ seconds. The graph is truncated at $t = 600$ seconds. In summary, Figure 5(a) shows that Rent-A-Server is able to dynamically recruit resources in response to increased client demands. As clients increase server load over time, Rent-A-Server is able to dynamically recruit needed surrogates to maintain relatively steady quality of service, delivering 800 ms average latency.

While the number of active HTTP servers varies from between one and eight, on average 5.7 servers are active. To contrast the performance of Rent-A-Server with static server allocation, the experiment is executed with identical parameters, with the exception that 6 fixed servers are allocated for the duration. Figure 5(b) depicts the results. Between $t = 0$ and $t = 120$ with relatively light client demand, static allocation outperforms Rent-A-Server, delivering an average latency of approximately 600 ms. However, it can be argued that resources are wasted because measurements indicate that three servers could deliver the same performance for eight clients. When all 32 clients are running between $t = 360$ and $t = 600$, Rent-A-Server's ability to dynamically recruit resources results in improved performance. During this time period, clients see 850 ms



(a) Rent-A-Server

(b) Fixed Server

Figure 5: Rent-A-Server Performance. The graphs plot average client latency as a function of time for the operation of retrieving a 2.5 KB HTML file over HTTP. In the Rent-A-Server graph, dark columns correspond to spawning of Rent-A-Servers onto surrogates while the lighter lines mark the start of a new group of 8 clients. The same experiment is run for the Fixed Server graph with static allocation of 6 HTTP servers.

average latency while the statically allocated resources become constrained, delivering 965 ms average latency. Thus, Rent-A-Server outperforms static resource allocation even when the average amount of consumed service resources stays constant.

The performance of Rent-A-Server demonstrates the power of dynamically recruiting resources for wide area services. However, it is equally important to provide a convenient interface for application development. Our implementation of Rent-A-Server in WebOS consists solely of the load daemon and additions to the Apache HTTP server to transmit state information to the load daemon and to transmit aggregate service state (in HTTP headers) to Smart Clients. The load daemon consists of 1000 lines of C++ code, and we added 150 lines of C code to Apache. Beginning with the WebOS framework, our prototype of Rent-A-Server was operational in less than one week.

9 Related Work

A number of recent efforts exploit computational resources available on the Internet for wide area parallel programming, including Wax [Stout 1994], Legion [Grimshaw et al. 1995], Atlas [Baldeschwieler et al. 1996], Globus [Foster & Kesselman 1996], and NetSolve [Casanova & Dongarra 1996]. WebOS shares a need for similar underlying technology with these systems (such as the need for a global name space and file system). However, these systems focus on a programming model for computing across the wide area, while our work focuses on system level support for building and running wide area applications.

Our work draws upon a large body of previous work in file systems exporting a global namespace, including AFS [Howard et al. 1988], Alex [Cate 1992], Coda [Kistler & Satyanarayanan 1992], Bayou [Terry et al. 1995], WebNFS [Sun Microsystems 1996], and UFO [Alexandrov et al. 1997]. The main differentiating point between WebFS and these earlier works is backward compatibility with the HTTP name space and a security model appropriate for wide area access. We plan to build on the work done in Coda and Bayou to address issues of replication and fault tolerance in the wide area.

Harvest [Chankhunthod et al. 1996], Squid [Squ 1996], and other Web caching efforts have focused on methods of extending the client cache across the Internet to caching proxies. Caching proxies in general are limited by a number of ways. Proxies are unable to produce dynamic Web content (i.e. the results of cgi-bin programs). Further, proxies are logical extensions of the client making it difficult for service providers to track such things as hit counts. Rent-A-Server addresses the limitations of proxy caching mechanisms by allowing full replication of overloaded services at locations determined by client access patterns.

The V kernel [Cheriton 1988] uses multicast for client communication to multiple members of a server group for load balancing and fault tolerance. This mechanism is related to our use of Smart Clients for extending service functionality onto the client. However, Smart Clients allow service-specific naming and load balancing algorithms. For example, the quality of the network fabric is non-uniform in the wide area, mak-

ing it important to distinguish sites based on the client's latency to each of the sites. Further, our approach enables migration of more general service functionality as demonstrated by the Internet Weather application described in Section 7.4.

The Active Networks proposal is to modify Internet routers to be dynamically programmable, either at the connection or packet level [Tennenhouse & Wetherall 1996]. The goal is to make it easier to extend network protocols to provide new services, such as minimizing network bandwidth consumed by multicast video streams. As in our work, a major motivation is to move computation into the Internet to minimize network latency and congestion. WebOS can be seen as a logical extension of Active Networks, where the active computing elements in the Internet can be servers in addition to the individual processors inside of routers operating on packet streams.

10 Conclusions

In this paper, we have demonstrated the synergy available from exporting traditional operating system functionality to wide area applications. Our prototype implementation, WebOS, describes one possible organization of these system services. In this framework, we make the following contributions. First, we show that extending server functionality onto client machines allows for more flexible implementation of name resolution, load balancing, and fault tolerance. Second, by providing a file system abstraction combining communication and persistence, we simplify the implementation of a number of wide area applications. Next, we present a methodology for coherently caching program results through the file system, speeding the performance of applications which must repeatedly execute programs with common inputs. Finally, we demonstrate how Rent-A-Server, an application developed in our framework, both improves system performance and more efficiently utilizes system resources for Web server access.

Acknowledgments

Both the content and presentation of this paper has greatly benefited from many discussions with members of the UC Berkeley NOW project. In addition, we would like to specifically thank Eric Anderson, Remzi Arpaci-Dusseau, Doug Ghormley, Ken Goldberg, Steve Lumetta, Steve McCanne, John Ousterhout, Dave Patterson, and Marvin Theimer for their feedback on the ideas presented in this paper.

References

[Alexandrov et al. 1997] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman. “Ufo: A Personal Global File System Based on User-Level Extensions to the Operating System”. In *Proceedings of the 1997 USENIX Technical Conference*, Anaheim, CA, January 1997.

[Anderson et al. 1995] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. “Serverless Network File Systems”. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pp. 109–126, December 1995.

[Baldeschwieler et al. 1996] E. Baldeschwieler, R. Blumofe, and E. Brewer. “Atlas: An Infrastructure for Global Computing”. In *Proc. of the Seventh ACM SIGOPS European Workshop: Systems Support for Worldwide Applcations*, September 1996.

[Belani et al. 1998] E. Belani, A. Vahdat, T. Anderson, and M. Dahlin. “The CRISIS Wide Area Security Architecture”. In *Proceedings of the USENIX Security Symposium*, San Antonio, Texas, January 1998.

[Berners-Lee 1995] T. Berners-Lee. “Hypertext Transfer Protocol HTTP/1.0”, October 1995. HTTP Working Group Internet Draft.

[Birrell et al. 1986] A. Birrell, B. Lampson, R. Needham, and M.Schroeder. “Global authentication without global trust”. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, May 1986.

[Brewer 1997] E. Brewer. Personal Communication, March 1997.

[Brisco 1995] T. Brisco. “DNS Support for Load Balancing”, April 1995. Network Working Group RFC 1794.

[Casanova & Dongarra 1996] H. Casanova and J. Dongarra. “NetSolve: A Network Server for Solving Computational Science Problems”. In *Proceedings of Supercomputing '96*, November 1996.

[Cate 1992] V. Cate. “Alex – a Global Filesystem”. In *Proceedings of the 1992 USENIX File System Workshop*, pp. 1–12, May 1992.

[Chankhunthod et al. 1996] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. “A Hierarchical Internet Object Cache”. In *Proceedings of the 1996 USENIX Technical Conference*, January 1996.

[Cheriton 1988] D. R. Cheriton. “The V Distributed System”. In *Communications of the ACM*, pp. 314–333, March 1988.

[Con 1989] Consultation Committee, International Telephone and Telegraph, International Telecommunications Union. *The Directory–Authentication*

Framework, 1989. CCITT Recommendation X.509.

[Crispo & Lomas 1996] B. Crispo and M. Lomas. "A Certification Scheme for Electronic Commerce". In *Security Protocols International Workshop*, pp. 19–32, Cambridge UK, April 1996. Springer-Verlag LNCS series vol. 1189.

[Dahlin et al. 1994] M. Dahlin, R. Wang, T. Anderson, and D. Patterson. "Cooperative Caching: Using Remote Client Memory to Improve File System Performance". In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation*, pp. 267–280, November 14–17 1994.

[Deering 1991] S. E. Deering. "Multicast Routing in a Datagram Internetwork". PhD thesis, Stanford University, December 1991.

[Dias et al. 1996] D. Dias, W. Kish, R. Mukherjee, and R. Tewari. "A Scalable and Highly Available Web Server". In *Proceedings of COMPCON*, March 1996.

[Dig 1995] Digital Equipment Corporation. *Alta Vista*, 1995. `http://www.altavista.digital.com/`.

[Feeley et al. 1995] M. M. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath. "Implementing Global Memory Management in a Workstation Cluster". In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

[Foster & Kesselman 1996] I. Foster and C. Kesselman. "Globus: A Metacomputing Infrastructure Toolkit". In *Proc. Workshop on Environments and Tools*, 1996.

[Fox et al. 1997] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. "Cluster-Based Scalable Network Services". In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Saintt-Malo, France, October 1997.

[Freier et al. 1996] A. Freier, P. Karlton, and P. Kocher. *Secure Socket Layer*. Netscape, March 1996.

[Goldberg et al. 1996] I. Goldberg, D. Wagner, R. Thomas, and E. Brewer. "A Secure Environment for Untrusted Helper Applications". In *Proceedings of the Sixth USENIX Security Symposium*, July 1996.

[Gong 1997] L. Gong. "Java Security: Present and Near Future". *IEEE Micro*, 17(3):14–19, May/June 1997.

[Gosling & McGilton 1995] J. Gosling and H. McGilton. "The Java(tm) Language Environment: A White Paper". `http://java.dimensionx.com/whitePaper/java-whitepaper-1.html`, 1995.

[Gribble & Brewer 1997] S. D. Gribble and E. A. Brewer. "System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace". In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*, Monterey, California, December 1997.

[Grimshaw et al. 1995] A. Grimshaw, A. Nguyen-Tuong, and W. Wulf. "Campus-Wide Computing: Results Using Legion at the University of Virginia". Technical Report CS-95-19, University of Virginia, March 1995.

[Howard et al. 1988] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. "Scale and Performance in a Distributed File System". *ACM Transactions on Computer Systems*, 6(1):51–82, February 1988.

[Int 1997] *Internet Weather Report*, 1997. `http://www.internetweather.com/`.

[Katz et al. 1994] E. D. Katz, M. Butler, and R. McGrath. "A Scalable HTTP Server: The NCSA Prototype". In *First International Conference on the World-Wide Web*, April 1994.

[Kistler & Satyanarayanan 1992] J. J. Kistler and M. Satyanarayanan. "Disconnected Operation in the Coda File System". *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[Kleiman 1986] S. R. Kleiman. "Vnodes: An Architecture For Multiple File System Types in SUN UNIX". In *Proceedings of the 1986 USENIX Summer Technical Conference*, pp. 238–247, 1986.

[Lampson et al. 1991] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. "Authentication in Distributed Systems: Theory and Practice". In *The 13th ACM Symposium on Operating Systems Principles*, pp. 165–182, October 1991.

[Luotonen & Atlis 1994] A. Luotonen and K. Atlis. "World-Wide Web Proxies". In *First International Conference on the World-Wide Web*, April 1994.

[Mat 1996] Matrix Information and Directory Services, Inc. *MIDS Internet Weather Report*, 1996. See http://www2.mids.org/weather/index.html.

[Mitzenmacher 1996] M. Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD dissertation, University of California, Berkeley, 1996.

[Mockapetris & Dunlap 1988] P. Mockapetris and K. Dunlap. "Development of the Domain Name System". In *Proc. SIGCOMM 88*, volume 18, pp. 123–133, April 1988.

[Nelson et al. 1988] M. Nelson, B. Welch, and J. Ousterhout. "Caching in the Sprite Network File System". *ACM Transactions on Computer Systems*, 6(1):134–154, February 1988.

[Net 1994] Netscape Communications Corporation. *Netscape Navigator*, 1994. http://www.-netscape.com.

[Poi 1996] PointCast. *The PointCast Network*, 1996. http://www.pointcast.com.

[Rivest & Lampson] R. L. Rivest and B. Lampson. "SDSI–A Simple Distributed Security Infrastructure". http://theory.lcs.mit.edu/ cis/sdsi.html.

[Rutkowski 1995] A. Rutkowski. "Testimony Before the U.S. House of Representatives Committee on Science". Available as http://www.isoc.org/rutkowski/ht_hearing_html, July 26 1995.

[Sarkar & Hartman 1996] P. Sarkar and J. Hartman. "Efficient cooperative caching using hints". In *Operating Systems Design and Implementation*, pp. 35–46, October 1996.

[Sollins & Masinter 1994] K. Sollins and L. Masinter. "Functional Requirements for Uniform Resource Names". RFC 1737, December 1994.

[Squ 1996] *Squid Internet Object Cache*, 1996. http://squid.nlanr.net/Squid/.

[Stout 1994] P. D. Stout. *Wax: A Wide Area Computation System*. PhD dissertation, Carnegie Mellon University, 1994. CMU-CS-94-230.

[Sun Microsystems 1996] "WebNFS: The Filesystem for the World Wide Web". Technical report, Sun Microsystems, 1996. See http://www.sun.com/webnfs/wp-webnfs/.

[Tennenhouse & Wetherall 1996] D. Tennenhouse and D. Wetherall. "Towards an Active Network Architecture". In *ACM SIGCOMM Computer Communication Review*, pp. 5–18, April 1996.

[Terry et al. 1995] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System". In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pp. 172–183, December 1995.

[Veizades et al. 1997] J. Veizades, E. Guttman, C. E. Perkins, and S. Kaplan. "Service Location Protocol". RFC 2165, June 1997. http://ds.internic.net/rfc/rfc2165.txt.

[Waldspurger & Weihl 1994] C. A. Waldspurger and W. E. Weihl. "Lottery Scheduling: Flexible Proportional-Share Resource Management". In *Operating Systems Design and Implementation*, pp. 1–11, Novemeber 1994.

[Wallach et al. 1997] D. S. Wallach, D. Balfanz, D. Dean, and E. W. Felten. "Extensible Security Architectures for Java". In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pp. 116–128, Saint-Malo, France, October 1997.

[Walsh et al. 1985] D. Walsh, B. Lyon, G. Sager, J. M. Chang, D. Goldberg, S. Kleiman, T. Lyon, R. Sandberg, and P. Weiss. "Overview of the Sun Network File System". In *Proceedings of the 1985 USENIX Winter Conference*, pp. 117–124, January 1985.

[Wobber et al. 1993] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. "Authentication in the Taos Operating System". In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pp. 256–269, December 1993.

[Wolski 1997] R. Wolski. "Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service". In *Proceedings of the 6th High-Performance Distributed Computing Conference*, August 1997.

[Yoshikawa et al. 1997] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. "Using Smart Clients to Build Scalable Services". In *Proceedings of the USENIX Technical Conference*, January 1997.