WebOS: Operating System Services for Wide Area Applications*

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

Abstract

In this paper, we demonstrate the power of providing a common set of Operating System services to wide-area applications, including mechanisms for naming, 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, we are building a system, WebOS, that provides basic operating systems services needed to build applications that are geographically distributed, highly available, incrementally scalable, and dynamically reconfigurable. 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 Web 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 [12] or Netscape's download page [31], 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 [34].

To address these problems, we are building WebOS, a framework for supporting geographically distributed, highly available, incrementally scalable, and dynamically reconfiguring applications. WebOS includes mechanisms for global naming [48], persistent storage [43, 42], remote process execution, resource management, authentication and security [4]. 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 functionality 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. presidential election; our framework would enable those services to handle the demand through dynamic repli-

*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. Dahlin was also supported by a NSF CAREER award (CCR 9733742). 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

cation. Recently, there has been a push toward the distribution of active components in the network, through technologies such as Active Networks [40] and Java [19]. The goal of WebOS is to provide a framework to assist application developers in utilizing programmable and active network components.

In this paper, we present an overview of the WebOS effort and a snapshot of the current status of our implementation. WebOS is composed of a number of different components. In the space allowed, we can only briefly describe the main ideas behind individual components, each of which is described in detail elsewhere [48, 43, 4, 42]. We have completed prototypes of each component and are now working to integrate them. The end goal is to provide a platform for the development and deployment for wide-area applications. Toward this end, 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. Next, we describe a security infrastructure that facilitates the fine-grained transfer of rights across the wide area.

To demonstrate the utility of WebOS as a substrate for the development of wide-area applications, 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 is organized to discuss each component of WebOS in turn. In Section 2, we present an overview of WebOS system components, before delving into more detail in Sections 3-6, which describe: (i) Smart Clients and for fault tolerant, load balanced access to Web services, (ii) WebFS, a global cache coherent file system, (iii) authentication for secure access to global Web resources, and (iv) a process control model supporting secure program execution and mechanisms for resource allocation, . 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.

- *Global Naming:* 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, global naming 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, 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 [46, 30, 21, 25, 41, 2]. In addition, we argue for benefit of integrating the file system with application-controlled efficient wide-area communication [43, 42].

- *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 [4]. 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 [17] for link level security.

3 Naming

In this section, we discuss an abstraction for location independent dynamic naming that allows WebOS clients to 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 to a list of replicated service representatives. Next, a service selection 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 Internet naming and shows how WebOS addresses these limitations through the use of Smart Clients. Our discussion focuses on naming in the context of HTTP service accessed through URL's. We are currently in the process of generalizing these techniques to other domains through Active Names, pieces of code responsible for name resolution that can be flexibly applied at different points in the network. The flexibility of Active Names could be used to provide URN's [36], transcoding [15, 16], resource discovery, and other applications that may benefit from the availability of code to perform name translation at various points in a distributed system.

To motivate the problem of naming in WebOS, we will use the following simple example. A user wishes to access a replicated Web site (such as the Alta Vista search engine or the download page for the latest version of Netscape Communicator). Ideally, users employ a single name for the Web service and the system translates the name to the IP address of the replica that will provide the best service to the client. Ideally, such translation should account for a number of factors, such as: the current members of the replicated service, available performance and current load at each server, network connectivity, client location, and network congestion between the client and replicas. Of course, the exact algorithm weighing these factors needs to be application and service specific. Further, given the current Internet architecture, it is impossible to accurately determine all the listed considerations, so approximations must be utilized.

In our implementation, we attempt to approach the above ideal by loading application and server specific code into end clients to perform perform name translation. These Smart Clients [48] enable extensions of server functionality to be dynamically loaded onto the client machine. Java's [19] portability and availability in all major Internet browsers allow us to distribute these extensions as Java applets.

The Smart Client architecture is summarized in Figure 1.

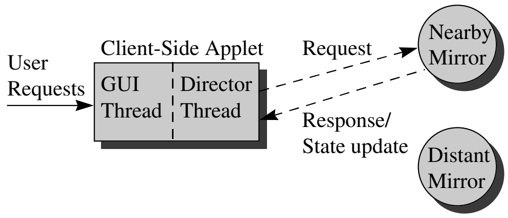


Figure 1. Two cooperating threads make up the Smart Client architecture. The GUI thread presents the service interface and passes user requests to the Director Thread. The Director is responsible for picking a service provider likely to provide best service to the user. The decision is made in a service-specific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

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. With 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 context of the Internet, clients using out of date information may make strictly worse choices than clients making random decisions [29]. 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.

Although the load balancing algorithm can be service specific, we implement the following algorithm by default. Service state, such as replicated group membership and load information, 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 use only static information.

As described, the Smart Client architecture provides a portable mechanism for fault tolerant and load balanced access to Web services. However, in a naive implementation, Java code must be retrieved each time a particular service is accessed, effectively doubling latency for small requests. In our implementation of Smart Clients, service-specific Java code is retrieved the first time a service is accessed and cached to disk using Java object serialization.

4 Persistent Shared State

WebOS seeks to raise the level of abstraction for large-scale distributed programs that manipulate shared state. Today, many such applications share state and transfer control using a network communication abstraction that relegates caching, cache consistency, security, transactional updates, and location-independence to application programmers. Following the analogy that it is often simpler to program parallel applications using shared memory as opposed to message passing, we contend that a global cache coherent file system abstraction will simplify the implementation of many applications. For example, the caching, cache consistency, and security provided by our prototype system, WebFS, greatly simplified the implementation of the Internet chat, remote compute engine, and Rent-A-Server applications described later in this paper.

Providing these abstractions to diverse applications in a wide-area network is challenging. For example, some applications, such as our remote compute engine and Rent-A-Server, require strong cache consistency while other applications, such as an Internet news service, may prefer weaker consistency to reduce overhead or to ensure that network failures do not delay updates. Therefore, a focus of our design is to provide flexibility so that demanding applications can control details of how key abstractions are implemented. We believe this approach is crucial for an Internet file system both because different applications have different demands and because the Internet's scale, limited performance, and unreliability can make it expensive to provide stronger guarantees than applications strictly require. WebFS associates a list of user-extensible properties with each file to extend basic properties such as owner and permissions, cache consistency policy, prefetching and cache replacement policy, and encryption policy. These properties are set and accessed through the UNIX `ioctl` system call.

Currently, WebFS implements the last writer wins [21] cache consistency protocol to support traditional file access as well as an IP multicast-based [11] 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 [32]. 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 [25] to deal with the disconnected operation endemic to today's Internet.

The WebFS prototype provides caching, cache consistency, and access-control. We implemented it as a dynamically loadable Solaris file system extension interfacing at the vnode layer [26]. The vnode layer makes upcalls to a user level WebFS daemon for file accesses not cached in virtual memory. WebFS uses a URL-based name space, and the WebFS daemon uses HTTP for access to standard Web sites. Thus, WebFS allows unmodified UNIX programs to take URL's (and URL's in a pathname syntax) in place of conventional file names (e.g., `ls /http/www-c.mcs.anl.gov/hpdc7/`). We chose to use URL's as the global namespace and HTTP for transport 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 [36] support for WebFS file names. Additionally, if the server site is also running WebFS, then authenticated read/write access and cache consistency are enabled through our own custom extensions to HTTP.

WebFS has been in day to day use for approximately one year by twenty users at the authors' site. It is publically available for download and has been successfully installed by a number of users un-affiliated with the authors. 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 versus cached access through NFS [46] is virtually identical.

5 Security and Authentication

Applications operating across the wide area are susceptible to a variety of potential attacks by sophisticated adversaries. To motivate the need for a wide-area security system, consider the simple example of a user wishing to run a simulation. Typically, if the simulation were executed locally, the program would run with all the user's privileges. When running the same simulation remotely, however, it is necessary to assign to the program the least set of privileges necessary to complete its task (e.g., read access to an input file and write access to an output file). This confinement of privileges protects users if the remote machine is compromised; while the simulation data may be usurped, the user's identity and other private files will hopefully remain secure. To provide this level of protection, a wide-area security system must provide fine-grained transfer of rights between principals in different administrative domains. The goal of our security abstraction is to transparently enable such rights transfer. CRISIS, the security system of WebOS, is described in another publication [4]; we present an overview here.

5.1 Validating and Revoking Statements

A principal contribution of CRISIS is the introduction of *transfer certificates*, lightweight and revocable capabilities used to support the fine-grained transfer of rights. Transfer certificates are signed statements granting a subset of the signing principal's privileges to a target principal. Transfer certificates can be chained together for recursive transfer of privileges. The entire chain of transfers is presented to reference monitors for validation, allowing for confinement of rights (e.g., a reference monitor can reject access if any principal in a chain of transfers is not trusted). Transfer certificates form the basis of wide-area rights transfer in WebOS, supporting operations such as delegation, privilege confinement, and the creation of roles (as described below).

All CRISIS certificates must be signed and countersigned by authorities trusted by both endpoints of a communication channel. A Certification Authority (CA) generates *Identity Certificates*, mapping public keys to principals. In CRISIS, CA's sign all identity certificates with a long timeout (usually weeks) and identify a locally trusted online agent (OLA) that is responsible for counter-signing the identity certificate and all transfer certificates originating with that principal. The endorsement period of the countersignature is application-specific, but typically on the order of 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 [9], 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.

Transfer certificates can be revoked modulo a timeout. 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.To revoke a particular privilege, the OLA that 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.

5.2 Processes and Roles

Given the ability 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 local resources and for mapping privileges 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. Restricting the rights available to a process is further detailed in Section 6.

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. It is often convenient to associate names with a specific subset of a principal's privileges. This functionality is achieved in CRISIS through named *roles*. 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 certificates. Thus, in creating new roles, principals act as their own certification authority [33]. The principal stores the role identity certificate and role transfer certificate in a *purse* of certificates that contains all roles associated with the principal.

5.3 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 principals 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 un-expired 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.

6 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. These virtual machines interact with the CRISIS security system to enforce rights restriction associated with different security domains. Thus, processes will be granted variable access to local resources through the virtual machine depending on the privileges of the user originally responsible for creating the process.

We use Janus [18] 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 [45] might be used to more flexibly allocate physical memory pages.

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 present 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) file appends and reads abstract away the need 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 [10] 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 [7]. 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 [2, 13], hints [35], 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 [28, 22, 47]. 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, 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). The goal of the implementation of Rent-A-Server is to demonstrate the power of using a unified system interface to wide-area resources and of moving a service out across the Internet.

8.1 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 [23]. 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

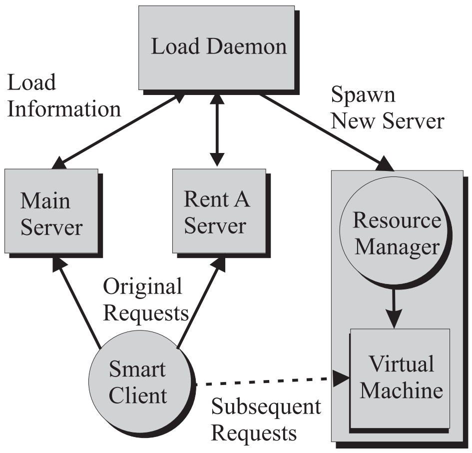


Figure 2. Rent-A-Server Architecture. HTTP servers periodically send load information to a load daemon. In response to an update, the load daemon transmits the state of all servers. In turn, the HTTP servers transmit this state information as part of the HTTP header to Smart Clients. The Smart Clients 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.

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 (another paper [42] describes our approach for relaxing this limitation). 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 [27]. In effect, the proxies are reducing generated advertising revenues by hiding page access counts.

8.2 System Design

In this subsection, we demonstrate how WebOS services simplify the implementation of this application. The architecture of the Rent-A-Server is described in Figure 2. Smart Clients are used for load balanced access to 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.

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*. 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 creates transfer certificates for the surrogate to access the WebFS files containing the executables (e.g. HTTP server) or internal service state (e.g. CGI scripts or internal database).

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

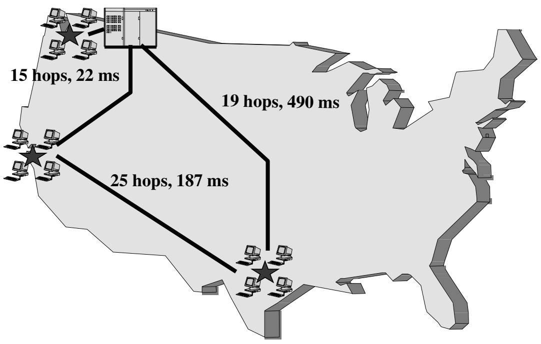


Figure 3. Rent-A-Server experimental setup. Clients at Seattle, Berkeley, and Austin act as clients of a service. For fixed server measurements, only a single server exists at the Seattle site. For Rent-A-Server measurements, the system begins with the single Seattle server, but additional servers are spawned at Berkeley and Austin in response to client load.

leases with a surrogate. The load daemon removes the surrogate from the appropriate ACL's.

8.3 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. While our measurements are preliminary and the system is not ready for production deployment, the measurements in this section suggest that further refinement of this model can potentially lead to improved wide-area Web service. Our experiments are conducted across the wide area as depicted in Figure 3. Eight sun Ultra workstations at each of Seattle, Berkeley, and Austin acting as clients of a Web service. Each client continuously requests the same 1 KB HTML file. Initially, there is a single HTTP server located in Seattle running Apache 1.2b6 on a Sun Ultra workstation. As described below, two surrogate Sun Ultra's are available at Berkeley and Austin to demonstrate the utility of Rent-A-Server. All the machines run Solaris 2.5.1.

Figure 3 also depicts the relative connectivity of the three sites, as measured by `traceroute` and `ping` on a weekend night. The reported numbers demonstrate best-case connectivity information. As shown in the figure, connectivity between Berkeley and Seattle is quite good, with only 22 ms round trip latencies reported by ping. Packets traveling from Berkeley to Austin have 187 ms latency, with approximately 2% of the packets dropped. Connectivity between Seattle and Austin is quite poor, with 490 ms latency and 20% of packets dropped.

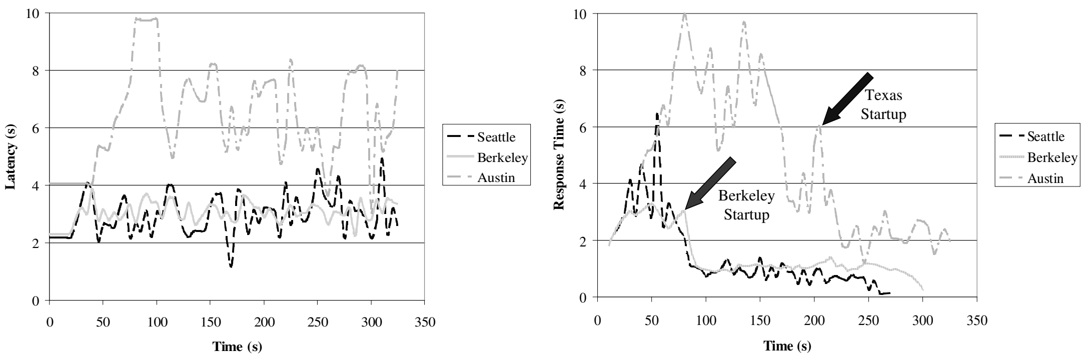
During the experiment, each client machine starts 8 Smart Client processes that continuously retrieve copies of the same 1 KB HTML file. The results of our tests are summarized in Figure 4. The graphs plot average client-perceived latency in seconds as a function of elapsed time, also in seconds. Figure 4(a) shows performance for the case where only a single server is available in Seattle. The graph shows that performance for clients at Berkeley and Seattle is quite poor, averaging approximately 3 seconds to retrieve the 1 KB HTML file from the Seattle server. Clients at Austin suffer from even worse performance, widely varying in average latency between 4 and 10 seconds. The poor performance of the Berkeley and Seattle results from an overloading of the single HTTP server in Seattle. The performance for the Austin clients relative to the Berkeley and Seattle clients is explained by the poor network connectivity between Austin and the HTTP server located in Seattle.

Figure 4(b) shows the improved performance available from using Rent-A-Server. In this case, approximately 90 seconds into the experiment, Rent-A-Server's load daemon spawns off an additional server at Berkeley. At this point, latency for both the Berkeley and Seattle improves into the .75 second latency range. Latency for the Texas clients is still poor because of the poor network connectivity between Texas and both Seattle and Berkeley. Thus, 200 seconds into the experiment a third server is spawned at Texas, with a corresponding improvement in latency for the Texas clients. What is not shown on this graph is the corresponding savings in wide-area bandwidth as clients at Berkeley and Seattle fall back to local servers as opposed to traversing wide-area links to reach the Seattle server.

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 [38], Legion [20], Atlas [3], Globus [14], Globe [44], and NetSolve [5]. A detailed comparison with the abstractions presented here and these projects is beyond the scope of this paper. However, WebOS



(a) Fixed Server (b) Rent-A-Server

Figure 4. Rent-A-Server Performance. The graphs plot average client latency as a function of time for the operation of retrieving a 1 KB HTML file over HTTP. Each line represents average latency sampled in 5 second intervals for clients located in Seattle, Berkeley, and Austin. In the Fixed Server graph, a single server in Seattle serves all pages. In the Rent-A-Server graph, a single server is initially available, but the system is able to spawn additional servers at Berkeley and Austin to improve client latency and reduce consumed wide-area bandwidth.

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 [21], Alex [6], Coda [25], Bayou [41], WebNFS [39], and UFO [1]. 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. Kermarrec et. al [24] propose a framework for supporting flexible cache consistency that is similar to the model we advocate.

Harvest [7], Squid [37], 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 [8] 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, making it important to distinguish sites based on the client's latency to each of the sites.

The Active Networks proposal is to modify Internet routers to be dynamically programmable, either at the connection or packet level [40]. 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 demonstrate 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 security system enabling the fine-grained transfer of rights across the wide area. To demonstrate the utility of these combined abstractions, we describe the implementation of a number of wide-area applications, including Rent-A-Server, an HTTP server capable of dynamically replicating itself across the Internet for both improved client latency and more efficient utilization of wide-area link bandwidth.

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

[1] 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.

[2] 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*, pages 109–126, Dec. 1995.

[3] 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.

[4] 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.

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

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

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

[8] D. R. Cheriton. The V Distributed System. In *Communications of the ACM*, pages 314–333, Mar. 1988.

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

[10] 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*, pages 267–280, November 14–17 1994.

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

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

[13] 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.

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

[15] A. Fox, S. Gribble, E. Brewer, and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, 1996.

[16] 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.

[17] A. Freier, P. Karlton, and P. Kocher. *Secure Socket Layer*. Netscape, Mar. 1996.

[18] 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.

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

[20] 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.

[21] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and Performance in a Distributed File System. *ACM Trans. Comput. Syst.*, 6(1):51–82, Feb. 1988.

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

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

[24] A.-M. Kermarrec, I. Kuz, M. van Steen, , and A. S. Tanenbaum. A Framework for Consistent, Replicated Web Objects. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, May 1998.

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

[26] S. R. Kleiman. Vnodes: An Architecture For Multiple File System Types in SUN UNIX. In *Proceedings of the*

1986 USENIX Summer Technical Conference, pages 238–247, 1986.

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

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

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

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

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

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

[33] R. L. Rivest and B. Lampson. SDSI–A Simple Distributed Security Infrastructure. `http://theory.lcs.mit.edu/ cis/sdsi.html`, 1996.

[34] 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.

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

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

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

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

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

[40] D. Tennenhouse and D. Wetherall. Towards an Active Network Architecture. In *ACM SIGCOMM Computer Communication Review*, pages 5–18, Apr. 1996.

[41] 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*, pages 172–183, Dec. 1995.

[42] A. Vahdat and T. Anderson. Transparent Result Caching. In *Proceedings of the 1998 USENIX Technical Conference*, New Orleans, Louisiana, June 1998.

[43] A. Vahdat, P. Eastham, and T. Anderson. WebFS: A Global Cache Coherent File System. See `http://www.cs.berkeley.edu/ vahdat/webfs/webfs.html`, December 1996.

[44] M. van Steen, P. Homburg, and A. S. Tanenbaum. The Architectural Design of Globe: A Wide-Area Distributed System. Technical Report Technical Report IR-422, Vrije Universiteit, March 1997.

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

[46] 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*, pages 117–124, Jan. 1985.

[47] 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.

[48] 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*, Jan. 1997.