WebOS: Operating System Services for Wide Area Applications

Amin Vahdat, Paul Eastham, Chad Yoshikawa, Eshwar Belani, Thomas Anderson, David Culler
Computer Science Division
University of California, Berkeley
and
Michael Dahlin
University of Texas, Austin

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, a global namespace, 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 wastes 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

Operating systems were originally developed to provide a set of common system services, such as I/O, communication, and persistent storage, to simplify application programming. With the advent of multiprogramming, this charter expanded to include abstracting shared resources so that they were as easy to use (and sometimes easier) as dedicated physical resources. The introduction of local area networks in the 80's expanded this role even further. A goal of network operating systems such as Locus [Popek et al. 1981], Mach [Accetta et al. 1986], Sprite [Nelson et al. 1988], Amoeba [Mullender et al. 1990], and V [Cheriton 1988] was to make remote resources over the LAN as easy to use as local resources, in the process simplifying the development of distributed applications.

With analogy to these systems, this paper argues that it is time to provide a common set of services for wide area applications, in effect to make wide area resources as easy to use as those on a LAN. The past few years has seen a dramatic increase in the number and variety of services that are available over the Internet, beyond simple documents to a wide variety of on-line services, such as search engines, real-time news, weather forecasts, interactive games and chat rooms, airplane reservations and ticketing, to name just a few.

Today, although the World Wide Web has made geographically distributed read-only data easy to use, geographically distributed computing resources are not. The result is that 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. This replication is managed *by hand* on both the server and the client side – users are forced to do manual polling between essentially equivalent services. 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 applications that are geographically distributed, highly available, incrementally scalable, and dynamically reconfiguring. WebOS includes mechanisms for resource discovery, a global namespace, 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 applica-

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. The source code for our system is publically available at `http://-now.cs.berkeley.edu/WebOS`.

tions 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 even all the way to the client. These 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 replication. Note that many popular Internet services already are statically set up to run at third party sites to pro-rate the cost of a fault tolerant, high bandwidth Internet connection over multiple services [Brewer 1997]; we are merely arguing that with the right support, these arrangements can be made dynamically.

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. 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.

Of course, the point can be made that the growth of the Internet can be partially attributed to its lack of central control, making the whole concept of an Internet operating system a contradiction in terms. Traditionally, Internet applications and protocols are developed and deployed without the intervention of any centralized authority. We are not arguing for strict central control. Rather, we believe that OS services should abstract the requirements of wide area applications. For example, TCP provides flow control, in-order, and reliable delivery over congested, unreliable medium without requiring each application programmer to understand exponential backoff, slow start, or round-trip time variance estimation [Jacobson 1988]. TCP's success is a testament to the ability to win widespread acceptance by providing a simple, convenient interface to network resources. We believe that system abstractions to other global resources can be adopted in a similar fashion, hiding the details of cache consistency, load balancing, fault tolerance, authentication, replication, resource allocation, and so on.

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 3-6, which describe: (i) Smart Clients for fault tolerant, load balanced access to Web services, (ii) WebFS, a global cache coherent filesystem, (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 discussion of future work in Section 10 and our conclusions in Section 11.

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, to make using geographically remote resources easier to use. 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, 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 address and URL's for the global name space. TCP is used to provide reliable communication, while SSL [Freier et al. 1996] is used 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. 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].

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 imple-

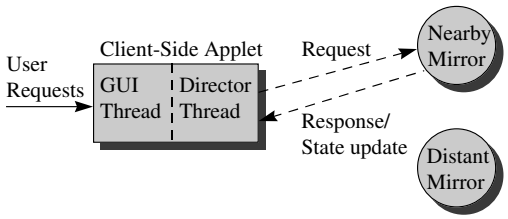


Figure 1: Two cooperating threads make up the Smart Client architecture. The GUI thread presents the service interface and passes user requests to the Director Thread. The Director is responsible for picking a service provider likely to provide best service to the user. The decision is made in a service-specific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

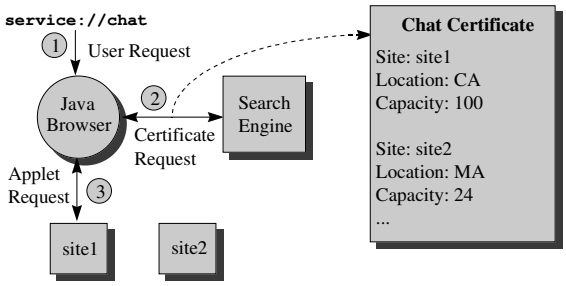


Figure 2: Bootstrapping applet retrieval in Smart Clients. A new service name space is introduced. In step 1, a name is translated into a certificate request from highly-available search engines in step 2. The certificate contains hints as to initial service group membership. The Smart Client applet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

ment 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 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 piggybacked 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 form `service://URL`; we leverage the URL name space to simplify integration with existing browsers. 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.

4 Global Namespace

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 filesystem for communication and synchronization. As empirical evidence, the implementation of chat (section 7.1) was simplified by the encapsulation of current conversations in secure globally accessible files. Combining caching with this interface has the added benefit of improving performance for applications accessing popular files. In this section, we discuss the implementation of our global file system and its integration with Transparent Result Caching (TREC), which allows the caching of dynamically generated Web content.

4.1 WebFS

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.cs.washington.edu/sosp16`). 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. Once a more scalable, fault tolerant naming mechanism, such as URNs [Sollins & Masinter 1994], is defined and widely deployed, it will be straightforward to incorporate the new mechanism into WebFS.

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.

Phase		NFS(s)	WebFS(s)
I	Create	1	1
II	Copy	7	7
III	Stat	3	10
IV	Scan	5	6
V	Compile	15	14
Total		31	38

Table 1: This table describes the performance of WebFS relative to the performance of NFS on the Modified Andrew Benchmark suite.

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 filesystem 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 reintegration [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.

To validate the performance of our system, we ran the modified Andrew Benchmark suite [Ousterhout 1990] for both WebFS and NFS [Walsh et al. 1985]. The NFS measurements were taken on a Sun Ultra Server making requests to an Auspex NFS server over Ethernet. For WebFS, a Sun Ultra Server made requests to a WebFS server running on a second Ultra Server also over Ethernet. The performance results are summarized in Table 1. Given the largely untuned nature of the code and the extra overhead associated with making upcalls to the user level, the performance of WebFS is reasonable for most phases of the benchmark. Significant slowdown is apparent in phase III, which executes a stat operation on 70 files. The factor of 3 slowdown is caused by the upcall that is made to the user level for every operation. We are in the process of modifying the code to store these values in the kernel when the file is initially copied, which should result in improved WebFS performance.

4.2 Transparent Result Caching

In developing WebFS, we observe that one fundamental obstacle to aggressive caching in the Internet is the widespread use of *dynamic objects*. These objects are generated on the fly by HTTP servers (e.g. through CGI-bin programs) and are marked uncacheable by both caching proxies and clients. Caching dynamic objects can result in a performance improvement when there is locality of reference and the objects are fairly expensive to generate. We have implemented *transparent result caching* (TREC) to enable caching of a certain class of dynamic objects. TREC allows program results to be stored as WebFS files, as in the Semantic File System [Gifford et al. 1991]. After the initial request for a particular program result, WebOS is able to return the cached contents of the file rather than re-executing the program.

Using the Solaris `/proc` filesystem, TREC generates dependency graphs of target programs by recording the name, environment, parent process, child processes, input, and output files. Once an object's profile is complete (the process and its children exit), TREC commits the profile to persistent storage and then executes a blocking WebFS system call to put "watchpoints" on both the program executable and its input files [Bershad & Pinkerton 1988]. The system call returns on modification to any of the target files, allowing TREC to inform WebFS of the change. On notification, WebFS unlinks the dynamic object results and sends invalidation messages to all sites caching the object.

Of course, TREC cannot cache all classes of dynamic objects. For example, programs may base results on random numbers or interaction with remote servers, making program results uncacheable. We use a WebFS file property to allow administrators to mark such program results as uncacheable. However, we believe that a significant class of Web dynamic objects can be cached using TREC. Further, if sites using TREC demonstrate performance improvements, structuring dynamic objects to be "TREC compliant" (cacheable) is straightforward. For example, a news service might generate its "front page" as HTML referencing the most timely article blurbs. The front page can be cached by remote proxies, and invalidated when the front page changes. This front page can be cached at both clients and proxies, potentially resulting in reductions in both server load and Internet congestion.

Our measurements of TREC show a uniform 30% overhead in performance for three benchmarks executed on an Ultra Server running Solaris 2.5.1: the modified Andrew Benchmark, compilation of 4000 lines of C code, and running latex on a 20 page document. The slowdown is caused by the multiple context switches associated with bouncing necessary system calls to TREC. We believe that this overhead can be largely eliminated by integrating TREC with an in-kernel system call interposition tool [Jones 1993, Bershad et al. 1995, Ghormley et al. 1996].

To demonstrate the utility of TREC, we take the common example of performing a query to an Internet search engine such as Alta Vista or HotBot [Brewer & Gauthier 1995]. Queries returning a large number of results are by default truncated to display the first 10-20 matches. However, if the next n matches are later requested, the entire query is re-executed [Brewer 1997]. Using TREC to cache search results for a short period of time can improve search overhead. To demonstrate this point, we measured the overhead of executing a simple CGI-bin program that returns 10 KB of HTML. For the Apache HTTP server, this operation took approximately 200 ms between 2 Ultra Servers on shared Ethernet. After modifying apache to check for TREC-generated program results before executing CGI bin programs, the same operation averaged 110 ms. The performance improvement results from avoiding the `fork` and `exec` operations Apache executes for CGI programs. The performance improvement assumes a no-op CGI program; the improvements would be more dramatic if the CGI program did real work. To further investigate this, we plan to take TREC measurements for search queries to an index of our local site produced by Excite [Exc 1997], a publically available indexing tool and search engine.

We note that TREC may also be useful outside of the context of WebOS. For example, it can be used to build a transparent make utility: by observing a program compilation once, TREC can generate the appropriate dependencies to automatically recompile program exe-

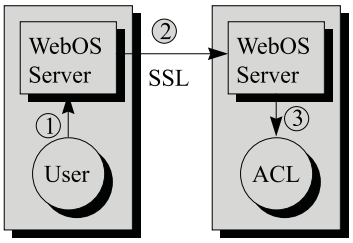


Figure 3: WebOS Security Model. First, users transfer some portion of their access rights to a WebOS server. When attempting to access a remote resource, WebOS servers communicate using SSL (step 2). Finally, the remote server checks if the proper credentials are held in local ACL's.

cutables (perhaps with a user prompt) if any of the input files change, including any nested ones, such as include files, or implicit dependencies, such as the executables for the compiler and assembler. As another example, if TREC were used to profile all local programs, it would allow users to query the file system for the origins of a given data file (e.g. determine the program and inputs used to generate the file in question). However, the use and implementation of TREC in such contexts is beyond the scope of this paper.

5 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 paranoia for programs that require it. In this section, will describe such a a model and show how it is used for secure, authenticated access to global resources.

WebOS provides security at a number of levels. First, two principals communicating at the link layer believe one another's identity and trust that the data cannot be compromised by a third party. Next, principals are able to have fine-grained control over which capabilities are transferred to remote WebOS processes running on their behalf. Finally, WebOS provides an interface for registering users and for specifying access rights to individual system resources. The WebOS security model is summarized in Figure 3.

5.1 Link Layer

WebOS relies upon a hierarchy of certification authorities (CA's). These authorities are responsible for producing X.509 certificates [CCITT 1988] mapping principals to their public keys [Diffie & Hellman 1977, Rivest et al. 1978]. These certificates are generated by the CA and contain the following information: the identity of the CA, the name of the principal, its public key, the period during which the mapping is valid, and a digital signature of the entire certificate by the CA. By using these certificates and by placing trust in the CA's, we are able to guarantee that any data successfully decrypted by a given public key was in fact generated by the associated principal.

We use the Secure Socket Layer (SSL) [Freier et al. 1996] to authenticate the identity of communicating servers to one another. SSL uses the public keys from a WebOS server's X.509 certificate to establish an RC4 symmetric key for session communication, and an MD5 hash of chunks of the byte stream to allow the principals to determine if an adversary is attempting to patch their communication. Symmetric keys are used for encryption because of their computational simplicity relative to public key cryptography. WebOS manages a cache of available connections to remote servers to avoid the overhead of establishing a secure connection for each logical operation.

5.2 Transfer of Rights

Principals register with WebOS and are able to control the transfer of portions of their rights to servers acting on their behalf [Burrows et al. 1989, Lampson et al. 1991, Wobber et al. 1993]. For example, a principal might register with a remote WebOS compute engine, transferring to the server the right to access a particular sub-directory containing the relevant data files. The server would be able to access the files necessary to carry out any computation on the principal's behalf but would be prevented from accessing the principal's private financial information located in another directory. A WebOS server can recursively transfer rights to a second server by signing the appropriate transfer certificate, unless a transfer right is not granted by the user.

In WebOS, principals transfer rights by creating a *transfer certificate* that specifies the resources in question (for example, all files in a particular WebFS subdirectory), the identity of the target WebOS server, and a time period during which the transfer certificate is valid. This transfer certificate is attached to the principal's X.509 certificate (identifying the principal) and an MD5 hash [Rivest 1992] of both the X.509 and transfer certificates encrypted in the user's private key. This en-

crypted hash certifies that the user actually created the transfer certificate. Using this scheme, a WebOS server is able to prove to other sites that it is authorized to act on the principal's behalf when accessing the resources specified in the transfer certificate. Note that the program generating the transfer certificate requires the principal's private key; however, this registration program is distributed as source and runs on a principal's local machine.

5.3 Specifying and Validating Rights

WebOS access rights are specified through Access Control Lists (ACL's). These access rights specify the list of principals which possess read, write, modify, and execute permission on a given resource. When a principal attempts to access a resource, WebOS sends the request with the proper transfer certificate to the site that owns the resource. The destination site takes the following steps in validating access to the resource in question: (i) it checks for a valid timestamp in the transfer certificate, (ii) it determines if the requesting WebOS server has the proper authority to act on the principal's behalf for the requested resource, as described in the transfer certificate, and (iii) it checks its local ACL's to determine whether the principal has the proper access permissions for the requested resource.

We chose to use ACL's because of their relative conceptual simplicity both from the principal's and implementor's point of view. A capability based system is more manageable in a global context where the access rights of potentially millions of users must be monitored. However, capabilities present complications for rights revocation and transfer. To partially address the issues of scale associated with ACL's, WebOS allows for global group access rights and implements an efficient storage/lookup mechanism for its ACL's. Based on the assumption that many resources will have identical groups of principals associated with them, WebFS resources point to unique "buckets" containing the appropriate rights specification. Lookups are executed by performing a hash of the requesting principal into the buckets.

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.

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` filesystem 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.

7 WebOS Applications

This section provides an overview of four applications built using the WebOS framework. The applications demonstrate that WebOS services enable and simplify their implementation. 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 modeled 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 file. As an added benefit, the WebFS interface becomes available for similarly structured 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 muilticasts 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]. 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 vs. 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.

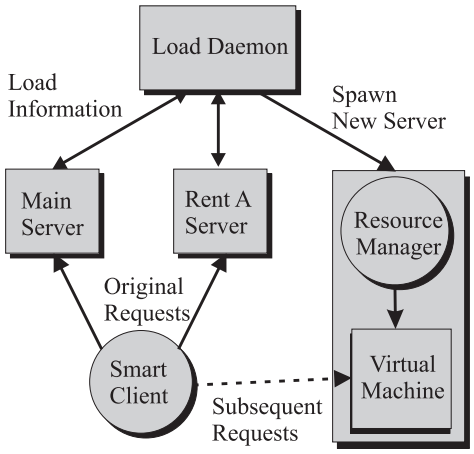


Figure 4: Rent-A-Server Architecture. HTTP servers periodically send load information to a load daemon. In response to an update, the load daemon transmits the state of all servers. In turn, the HTTP servers transmit this state information as part of the HTTP header to Smart Clients. The Smart Clients 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.

8.3 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 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).

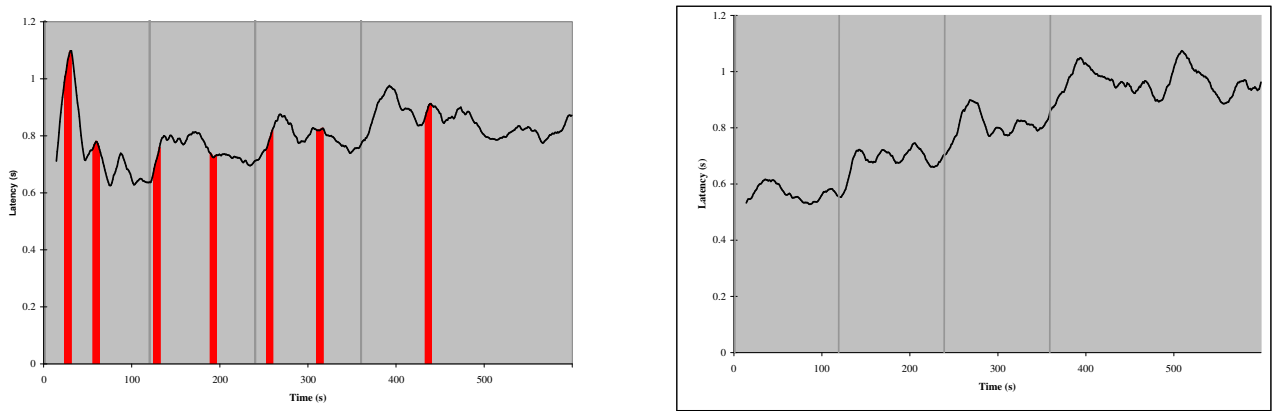
Each Rent-A-Server maintains information about client geographic locations (location is sent by Smart Clients as part of the HTTP request) and its own load information in the form of requests per second and bytes transmitted per second. Each Rent-A-Server periodically transmits this state information 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 piggybacked 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 certificate (with an expiration date) 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.

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 be-



(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.

half 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) 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 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], 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, making 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 recent 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 Future Work

The goal of our work is to demonstrate that providing operating system services across the wide area will both simplify application development and more efficiently manage global resources. However, the WebOS prototype is only a single point in a large space of possible design alternatives. Preliminary results from the applications we have built demonstrate that WebOS is capable of simplifying application development and of efficient resource utilization. However, many questions remain before determining the "correct" organization for a wide area operating system.

To explore this design space, it is necessary to deploy our system in the wide area. While we believe our fundamental design decisions to be sound, their ramifications and their impact on application performance can only be determined when taking measurements in real-world situations. To this end, we are making our software publically available; a number of sites have already agreed to participate in our wide area testbed.

Using this testbed, we plan to determine filesystem consistency requirements in the face of host/network failure. We will begin with the Coda model of optimistic concurrency. Another avenue we will explore is Smart Client load balancing algorithms that efficiently utilize network resources (e.g. bandwidth), minimize server load, and deliver the highest performance to the end clients. We also plan to explore placement policies of dynamically spawned Rent-A-Servers when clients

are geographically distributed. Finally, our implementation is currently restricted to Solaris machines. To truly support wide area applications, WebOS services must be ported to support heterogenous platforms.

11 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

[Accetta et al. 1986] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. "Mach: A New Kernel Foundation For UNIX Development". In *Proceedings of the 1986 USENIX Summer Conference*, pp. 93–112, June 1986.

[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.

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

[Bershad & Pinkerton 1988] B. N. Bershad and C. B. Pinkerton. "Watchdogs — Extending the UNIX File System". *Computing Systems*, 1(2):169–188, Spring 1988.

[Bershad et al. 1995] B. N. Bershad, S. Savage, E. G. S. P. Pardyak, M. Fiuczynski, D. Becker, C. Chambers, and S. Eggers. "Extensibility, Safety and Performance in the SPIN Operating System". In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

[Brewer & Gauthier 1995] E. Brewer and P. Gauthier. "The Inktomi Search Engine". http://-www.inktomi.com, 1995.

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

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

[Burrows et al. 1989] M. Burrows, M. Abadi, and R. Needham. "A Logic of Authentication". In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pp. 1–13, December 1989.

[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.

[CCITT 1988] CCITT. "The Directory — Authentication Framework". Recommendation X.509, 1988.

[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.

[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.

[Diffie & Hellman 1977] W. Diffie and M. Hellman. "New Directons in Cryptography". In *IEEE Transactions on Information Theory*, pp. 74–84, June 1977.

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

[Exc 1997] Excite Corporation. *Excite for Web Servers*, 1997. http://www.excite.com/navigate/AThelpdoc.html.

[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.

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

[Ghormley et al. 1996] D. P. Ghormley, D. Petrou, and T. E. Anderson. "SLIC: Secure Loadable Interposition Code". May 1996. See `http://now.cs.berkeley.edu/Slic/`.

[Gifford et al. 1991] D. K. Gifford, P. Jouvelot, M. Sheldon, and J. O'Toole. "Semantic File Systems". In *13th ACM Symposium on Operating Systems Principles*, October 1991.

[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.

[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.

[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/.

[Jacobson 1988] V. Jacobson. "Congestion avoidance and control". In *Proceedings of ACM SIGCOMM '88*, pp. 314–329, August 1988.

[Jones 1993] M. B. Jones. "Interposition Agents: Transparently Interposing User Code at the System Interface". In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pp. 80–93, December 1993.

[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.

[Mullender et al. 1990] S. J. Mullender, G. van Rossum, A. S. Tanenbaum, R. van Renesse, and H. van Staveren. "Amoeba: A Distributed Operating System for the 1990s". *IEEE Computer Magazine*, 23(5):44–54, May 1990.

[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.

[Ousterhout 1990] J. Ousterhout. "Why Aren't Operating Systems Getting Faster As Fast As Hardware?". In *Proceedings of the 1990 Summer* USENIX *Conference*, pp. 247–256, Anaheim, CA, June 1990.

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

[Popek et al. 1981] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G. Thiel. "LOCUS: A Network Transparent, High Reliability Distributed System". In *Proceedings of the 8th ACM Symposium on Operating Systems Principles*, pp. 169–177, December 1981.

[Rivest 1992] R. L. Rivest. "The MD5 Message Digest Algorithm". RFC 1321, April 1992.

[Rivest et al. 1978] R. L. Rivest, A. Shamir, and L. Adelman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In *Communications of the ACM*, volume 21, February 1978.

[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.

[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.

[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.

[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.

[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.