WebOS: Software Support for Scalable Web Services

Amin Vahdat, Michael Dahlin, Paul Eastham, Chad Yoshikawa,
Thomas Anderson, and David Culler
Computer Science Division
University of California
Berkeley, CA 94720

Abstract

The burgeoning popularity of the Web is pushing against the performance limits of the underlying infrastructure, presenting a number of difficult challenges for the Web *as a system*. We believe that resources such as connectivity, storage, computation, latency, and bandwidth are likely to remain constrained in the future. Thus, we are building a higher level Web operating system to efficiently manage these resources for the benefit of all Web users. Our prototype, WebOS, is designed to support highly available, incrementally scalable, self-tuning, dynamically reconfiguring and geographically aware Web services. WebOS includes mechanisms for naming Web services, coherent data replication, efficient distribution of client requests across the wide area, safe execution of arbitrary executables on remote sites, and authentication for secure access to Web resources.

1 Introduction

The growth rate and scale of the Web presents a family of difficult challenges for the Web *as a system*. While the underlying communications infrastructure has proved surprisingly robust, we believe that the presence of a higher level system design will enable advances in the capabilities of Web applications. In short, we believe the development of an operating system for the Web will provide Web applications convenient access to shared resources. The operating system will efficiently manage these resources for the benefit of all users of the system.

Today, the operating environment of the Web could be characterized as location-unaware browsers making mostly read-only requests to hardware-constrained servers over an almost oblivious network fabric. Our goal is to demonstrate a different vision of the Web operating environment where computation and storage servers in the Web function cooperatively to provide efficient access to resources supporting Web applications. These core servers are incrementally scalable and highly available, browsers are aware of the service characteristics and the state of the network connection to these servers, and servers can cooperate to share load and distribute data throughout the Web to both reduce latency and save bandwidth. This vision requires a software framework for development of advanced Web applications.

To demonstrate that our vision of the Web is achievable, we are constructing WebOS, a framework for building highly available, incrementally scalable, self-tuning, dynamically reconfiguring and geographically aware Web services. WebOS must provide traditional local OS functionality at the global level, including: naming, resource allocation, inter-process communication, scheduling, fault tolerance, and authentication. To this end, we have initial prototypes of: (i) Smart Clients for fault tolerant, load balanced access to Web services, (ii) WebFS, a global cache coherent filesystem, (iii) a virtual machine supporting secure program execution and mechanisms for resource allocation, and (iv) authentication for secure access to global Web resources. To evaluate WebOS, we are developing a global cooperative cache of Web pages, allowing replication and migration of Web services during times of peak demand.

The rest of this paper will describe the WebOS design, implementation status, and supported applications. Section 2 describes our requirements for high-performance scalable Web services. Section 3 and 4 presents the WebOS design and implementation status. Section 5 details an application we are building to evaluate WebOS. Section 6 describes related work

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 0401156), California MICRO, the AT&T Foundation, Digital Equipment Corporation, Exabyte, Hewlett Packard, Intel, IBM, Microsoft, Mitsubishi, Siemens Corporation, Sun Microsystems, and Xerox. Anderson was also supported by a National Science Foundation Presidential Faculty Fellowship. Yoshikawa is supported by a National Science Foundation Fellowship. The authors can be contacted at {vahdat, dahlin, eastham, chad, tea, culler}@cs.berkeley.edu.

and Section 7 concludes.

2 Web Service Requirements

In this section, we outline some of the requirements for a software framework supporting high performance scalable Web services.

- *End-to-end continuous operation*: With millions of potential clients geographically distributed around the world, someone somewhere is always requesting service. Computer systems providing continuous service over a period of years must not only mask failures, but must also demonstrate end-to-end availability based on client perceptions. It does not matter if the service is "working" if the client cannot access it because of router or DNS problems. As a motivating example, one study [44] found that the probability of encountering a major Internet routing pathology was between 1.5% and 3.4%, and that the failure rate may increase with increasing Internet use.

- *Cheap incremental scalability*: To make it easy to create new services, it is crucial to minimize the initial hardware and software investment required to go on-line. It is equally crucial for the service to be able to add hardware resources as the service becomes more popular.

- *Graceful burst behavior:* The aggregate behavior of millions of clients is highly bursty; for example, the USGS web site was effectively unusable for hours after a recent (small) Northern California earthquake [50]. Bursty client behavior requires server software to be able dynamically recruit resources over the Internet. In addition, burstiness puts a premium on adapting in advance—using idle time to reconfigure the system to better handle any impending spikes in demand. For example, the IRS Web site becomes extremely loaded every year in mid-April.

- *Geographically dispersed resources*: The Internet is slow, costly, and prone to congestion. Latency (often best predicted by the number of network hops between client and server) is often the most important metric for measuring performance delivered to the client. Further, the current (hidden) cost of Internet traffic is a few dollars per terabyte-mile [25]. These considerations suggest that to both reduce latency and consume less bandwidth, service providers should move portions of their service across the Internet. Once a service becomes sufficiently popular, it is almost forced to become geographically distributed; no single point of connection on the Internet is capable of servicing simultaneous requests from millions of users all over the world.

Existing Web services currently address these requirements on an ad-hoc and incomplete basis. WebOS is designed to support construction services capable of dynamic geographic migration and replication based on client demands. Any necessary service state is replicated on demand through a global cache coherent file system. Service code such as HTTP servers or CGI-bin programs are executed in a protected environment. Authentication mechanisms are provided to ensure secure access to private service state (e.g. customer database or Web index) and to safeguard against unauthorized execution of service programs. To allow the client to automatically choose the service provider able to deliver the best performance to the user, the client code is enhanced to become aware of service replication, its location relative to the nearest server, and the relative load of each of the servers. Similarly, clients can transmit their geographic location to the service so that intelligent migration/replication decisions can be made before a service becomes overloaded.

3 Scalable Web Service Access

This section and the next describe the design and implementation status of the WebOS prototype. The goal of the prototype is to support services running at multiple, geographically distributed sites across the Internet. One motivation is to optimize for Internet latency and congestion; by moving services closer to clients, we reduce response time and consume less Internet bandwidth. Another motivation is the burstiness of Web requests. We believe the appropriate model for effectively delivering Web services to end users is that of the power grid where underutilized resources can serve excess demand. For example, a sporadically popular web site, such as the IRS near April 15th, should buy only enough local hardware to handle the typical demand and dynamically recruit extra hardware resources over the Internet to handle peak loads.

3.1 Current Approaches

Today, some popular services such as the Alta Vista search engine [21] or the Netscape home page [43] are geographically distributed but 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 in the local area techniques such as HTTP redirect [6] or DNS Aliasing [11, 38] 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 adding a round trip message latency to each request. Further, the front-end machine serving redirects is both a single point of failure and a central bottleneck for very popular services. Finally, any load-balancing mechanisms must be re-implemented on the front-end on a service-by-service basis.

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 [19].

3.2 Smart Clients

In WebOS, we address the shortcomings of existing solutions for scalable service access through the Smart Client architecture [56]. Smart Clients allow service-specific extensions to be dynamically loaded into the client to provide tracking of the mobile service, load balancing among individual servers, and fault transparency to end users. The portability of Java and its availability in all major Internet browsers allow us to distribute these extensions as Java applets. A service-specific applet is retrieved by the user's browser each time a service is accessed. The applet contains hints for the locations of service providers. A typical applet's code is composed of two cooperating threads: a customizable graphical interface thread providing 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 mask transparency in the case of failure. Both the interface and director are extensible in a service-specific manner. A prototype of Smart Clients is currently operational and has been used to implement scalable access to Internet services, including FTP, telnet, and Internet chat.

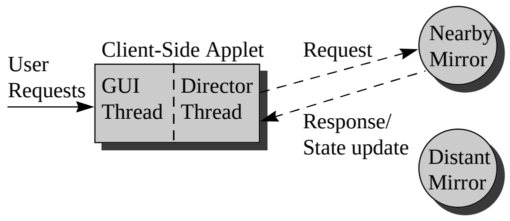

Figure 1: Two cooperating threads make up the Smart Client architecture. The GUI thread presents the service interface and passes user requests to the Director Thread. The Director is responsible for picking a service provider likely to provide best service to the user. The decision is made in a service-specific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

Figure 1 describes the Smart Clients architecture. A GUI thread presents the service interface and accepts requests which are in turn passed to the director thread. The director is responsible for choosing the individual server likely to provide the best possible performance to the user. This decision is made in a service-specific manner, using state such as server load, distance from the client, and past server performance. The director also maintains any information necessary to retransmit the request in case of server failure. As part of the reply, servers are able to transmit service update information to the Smart Client applet, such as additional service representatives or significant changes in server load.

3.3 Bootstrapping Applet Retrieval

While the Smart Client architecture provides a portable mechanism for fault tolerant, load balanced access to Web services, a number of issues remain to be addressed. Services must still be named through URLs, implying a single point of failure and a central bottleneck. Further, an applet must be downloaded each time the service is to be accessed, 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://chat`. The meta-applet translates these names into *service certificate* requests to well-known and highly available Internet search engines (such as Alta Vista). The certificate contains a hint as to initial service membership (individual

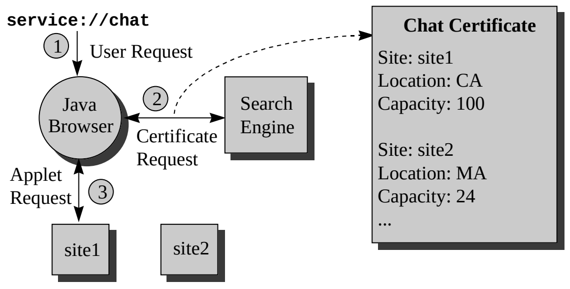

Figure 2: Bootstrapping applet retrieval in Smart Clients. A new service name space is introduced. A name is translated into a certificate request from highly-available search engines. The certificate contains hints as to initial service group membership. The Smart Client applet is retrieved from one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

server machines). The service's Smart Client applet is downloaded from one of these sites and operation proceeds normally as described in Section 3.2. Both the service certificate and Smart Client applet are cached to disk by the meta-applet to avoid this bootstrapping mechanism on subsequent accesses. Further, Java object serialization [36] is used to cache the Smart Client applet, allowing persistent service state between successive accesses.

4 Moving Services Around the Internet

The Smart Clients architecture provides a model for moving some service functionality onto the client machine. Further, common solutions to naming, load balancing, and fault tolerance are packaged in the Smart Client applets. In this section, we describe WebOS support for services capable of dynamically growing across the Internet to best support client demands. In our vision, machines across the Internet will be available (perhaps for compensation) to host, either permanently or temporarily, expansion of popular Internet services. To support this model, the WebOS design provides the following components: (i) a global cache coherent file system to enable services to name, access, and modify shared state, (ii) a web virtual machine to safely execute any programs needed by the service, and (iii) an authentication model allowing secure access to private service state (data and executables).

4.1 Global Cache Coherent Filesystem

WebFS [54] is the initial prototype of a URL-based cache coherent file system. WebFS allows UNIX programs to take URLs (and URLs in a pathname syntax) in place of conventional file names (e.g., `lpr /http/www.cs.berkeley.edu/ps/*`). This global file system provides a common set of services to both clients and servers, allowing them to abstract the mechanics of moving data around the Internet. An application running on a remote machine can access data in the user's home file system and home applications can access remote data without cross-mounting file systems or explicitly dropping to a network transfer protocol, such as FTP or HTTP. Migrating services operate on both the home and remote cluster in the same global file space. Data is paged over to the remote site on demand, rather than being explicitly replicated from outside the application. In this respect, the global name space provided by WebFS provides a similar service as location-independent object references in Emerald [37]. With WebFS, spawning a service on a remote server is analogous to spawning a thread on another processor in an SMP, except that the inter-server interaction is through a shared global file space, rather than a shared memory address space, and it is an entire collection of service threads that are spawned.

A fundamental difference between WebFS and existing 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 that they can live with out-of-date data with manual revalidation. By contrast, we believe Web applications require complete and well-defined file system semantics. Since the choice of coherence semantics implies tradeoffs in performance, consistency, and availability, WebFS provides application-controllable and dynamically adaptable cache coherence policies. For example, strict consistency with optimistic reintegration could be used for files shared between two geographically distributed sites implementing a service [39, 52], 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. A number of cache coherence mechanisms have been proposed for distributed systems, such as leases [30], last writer wins [35], volume reintegration [41], and modified time to live [33]; we plan to evaluate these in the context of the workloads generated by Web services.

Another avenue we plan to explore is using IP Multicast [18, 23] to provide update-based cache coherence. For example, a news service could select multicast update policy for its widely shared, frequently updated files, such as vote counts on election night. Other Web services such as chat rooms and whiteboards have long been structured using multicast; we can simplify these applications by folding the multicast support into the file system (for these applications, we use weak coherence since updates need not arrive in a consistent order at all sites [7]). Both the last-writer wins and the multicast update/invalidate coherence policies have been implemented in WebFS [54].

We also plan to investigate *transparent result caching*. HTTP servers translate some URLs into requests to dynamically generate a Web page. HTTP servers translate URLs matching a certain pattern (i.e. having `cgi-bin` at a certain point in the path) into requests to run a program responsible for dynamically generating the contents of the page. Today, these pages cannot be cached since the generation of these objects is dependent on the program input (i.e. from an HTML form). We have prototyped a system that records the program, the environment, and inputs to such programs. These records are generated by intercepting the certain system calls using the Solaris `proc` filesystem interface. If any of the program inputs or the program itself are modified, the file system invalidates all cached copies of the file (the result of the program). As an example, a news service may maintain a database of news stories. Requests for the "front-page" might be translated into a query for abstracts of the most recent stories. Since the database may be updated frequently (i.e. if the news site is providing election results), any resulting HTML should not be cached by Web browsers. With transparent result caching however, the filesystem is able to track changes to the underlying database and is able to invalidate any cached copies of the query result once the underlying database is modified.

4.2 Web Virtual Machine

We believe that compute engines across the Internet, called *surrogate sites*, will be available to host the expansion of popular Web services or simply to act as compute engines. These surrogates may be recruited on a permanent basis to allow a service to better handle its average load, or on a temporary basis to handle temporal spikes in demand. Internet hosts can be motivated to provide surrogate service in exchange for compensation, monetary or otherwise. In our model, services provide a template for duplicating themselves on remote surrogates. Through the WebOS security model, the surrogate is given the necessary capabilities to start the appropriate programs (perhaps a custom HTTP server) and to access any necessary state (such as a customer database) through the global file system.

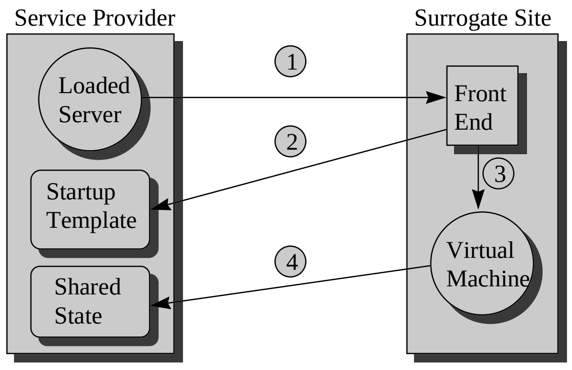

Figure 3: Surrogate host design. Overloaded services contact a front-end process on a surrogate host to request expansion. Once terms are negotiated, a startup template is accessed in step 2, allowing the front end to start necessary processes in a virtual machine in step 3. These processes can access and modify any shared service state through the global file system in step 4.

The surrogate system design is summarized in Figure 3. A front-end process on the surrogate is responsible for negotiating terms of setup. To ensure the system integrity of the surrogate and to ensure that executed processes do not interfere with other processes running locally, the front-end creates a *virtual machine* where all processes for the service execute. We are using the Secure Remote Helper Applications model [29] to create a virtual machine which executes processes with limited privileges, preventing them from interfering with the operation of processes in other virtual machines. Once processes are initialized in the virtual machine, the expanding service can inform Smart Clients of the location of a new service provider. In the absence of Smart Clients, traditional techniques such as HTTP redirect or DNS aliasing can be used to utilize the new surrogate site.

4.3 Security and Authentication

To fully support dynamically migrating services we need to provide secure authenticated access to public and private Web data. In WebOS, we leverage work in wide area distributed system authentication and access control [20, 46, 49, 27, 12, 55, 26] to build access control lists for access to files in the global name space. Individuals are uniquely identified by a public key, allowing for unique identities across organizational boundaries. We assume the presence of one or more trusted, replicated authorities (similar to DNS) which associate individuals with their public key. Using this model, WebFS files can be protected from unauthorized access. Further, WebOS authentication will be used as a basis for resource allocation policies in the Web virtual machine, allowing job priorities and access rights to be set on a per-user basis. A description of a prototype implementation of WebFS authentication is available [5].

5 Web Cooperative Cache

We are using WebOS to build a geographically distributed Web cooperative cache to both validate our design and to provide an immediate benefit to the Internet by doing more intelligent caching of Web content. Rather than caching pages only at a single server site or at the client, the Web cooperative cache will reposition and replicate the content of the Web to minimize latency while limiting consumed bandwidth and storage capacity. Because of Internet latency and congestion, performance will be improved if clients can exploit the contents of Web caches of geographically nearby servers. A number of challenges must be addressed in building this service: (i) naming of the site able to deliver the best performance from the client's perspective, (ii) allowing cache servers to replicate the execution environment necessary to create dynamic Web pages (i.e. created by CGI-bin programs), (iii) allowing end users to coherently cache these dynamic pages, and (iv) replicating and migrating Web services in response to client demands.

Individual WebOS components support construction of such a Web cooperative cache. In our design, a Smart Client applet will be responsible for retrieving URLs on behalf of the user. A well-known function is applied to each URL to determine the required service certificate for the URL group. Roughly speaking, each URL group will thus constitute an Internet service. At least one cache server is associated with each URL group and is included in the service certificate. The exact membership of the server group is discovered through interaction with the well-known server(s). Over time, the Smart Client applet is able to maintain relatively accurate server group membership for all URL groups that the user is interested in.

If pages in a single URL group become very popular, a strategically located surrogate machine is recruited to augment the overloaded group. Once the identity of this new new server is propagated to Smart Clients, part of the load for the URL group can be transferred to this new server. Normal WebFS caching mechanisms are used to demand page requested pages. WebFS cache coherence mechanisms make the chances of delivering stale pages to end users less likely than using current Web caching mechanisms. The Web Virtual Machine and WebOS authentication model are used to execute CGI-bin programs (or other custom executables) and to securely access any necessary state information (such as a customer database). Finally, transparent result caching is used to cache dynamic Web pages and to ensure invalidation of these dynamic pages at the proper time.

6 Related Work

We are able to leverage and evaluate a huge research literature in the context of Web services, including: models for building fault tolerant applications [4, 8, 45, 34, 9, 40, 7, 17, 31, 47, 10, 48]; process management across the local area [53, 22, 42, 28, 57, 14]; wide area file systems [35, 15, 39, 52, 16, 33, 1]; wide area security systems [20, 46, 49, 27, 12, 55, 26]; and mechanisms for hiding fault tolerance, load balancing, and dynamic relocation from end-users [31, 38, 6, 11, 2].

In this section, we focus our discussion on efforts to exploit computing and storage resources of the Internet. Perhaps the closest to our work is the recent Active Networks proposal [51]. Active Networks moves computing into the Internet by modifying Internet routers to be dynamically programmable, either at the connection or the packet level. 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 of Active Networks 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.

Finally, a number of recent efforts exploit computational resources available on the Internet for wide area parallel programming, including Atlas [3], Globus [24], Legion [32], and NetSolve [13]. Although we share a lot of underlying technology with these systems (such as the need for a global name space and file system), our emphasis and our solution is different. These systems all assume that wide area network performance will improve to the point where the world can be seen as a single, huge parallel computer. By contrast, our work is motivated by the observation that in practice wide area networks are slow and expensive, and are likely to remain so for the foreseeable future. Thus, we move computation around the Internet, not to exploit parallelism, but to minimize network latency and congestion in delivering services to clients. One implication is that web service providers will pay to install physically and logically secure high performance servers at geographically strategic Internet locations. As a result, we benefit from a simpler security model relative to these efforts, since we are not trying to exploit idle resources on millions insecure desktops.

7 Conclusions

The explosion in the use of the Internet has created a need for scalable, fault tolerant, and geographically distributed web services. Unfortunately, the technical understanding of how to build web services that can support very large numbers of clients has lagged far behind the demand. Our hypotheses are that services need to be able to migrate dynamically between geographically distributed sites to handle bursty requests and to avoid network congestion, and that clients have a role in providing transparent access to dynamically migrating services. The goal of WebOS is to provide a software framework for highly available, dynamically adaptable, and incrementally scalable web services running on a set of geographically distributed machines.

A number of key components comprise the WebOS architecture. Smart Clients allow portions of service functionality to be migrated onto the client machines. Smart Clients cooperate with servers to provide load balanced fault tolerant access to Web services. To aid in construction of scalable services, WebOS also includes a global cache coherent file system, an authentication model to ensure secure access to private state and executables, a virtual machine for safe execution of service binaries, and resource allocation tools to allow for adjudication among competing Web services. Most of the key WebOS components have been prototyped. To demonstrate the viability of our ideas and the prototype, we are building a Web cooperative cache of web pages, allowing surrogates located at strategic points in the Internet to serve cache-coherent content on behalf of heavily loaded or distant Web services. We believe that the WebOS infrastructure will also allow rapid prototyping of a number of other compelling scalable Internet services, including: a low-latency chat service, a distributed compute server, and multicast-based news broadcast applications.

References

[1] Albert D. Alexandrov, Maximilian Ibel, Klaus E. Schauser, and Chris 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] Eric Anderson, David Patterson, and Eric Brewer. The Magicrouter, an Application of Fast Packet Interposing. See http://HTTP.CS.Berkeley.EDU/˜eanders-/magicrouter/, May 1996.

[3] Eric Baldeschwieler, Robert Blumofe, and Eric Brewer. Atlas: An Infrastructure for Global Computing. In *Proc. of the Seventh ACM SIGOPS European Workshop: Systems Support for Worldwide Applcations*, September 1996.

[4] J. Bartlett. A NonStop Kernel. In *Proceedings of the 8th ACM Symposium on Operating Systems Principles*, pages 22–29, December 1981.

[5] Eshwar Belani, Alex Thornton, and Min Zhou. Security and Authentication in WebFS. See http://-now.cs.berkeley.edu/WebOS/security.ps, December 1996.

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

[7] Ken P. Birman. The Proecss Group Appraoch to Reliable Distributed Computing. *Communications of the ACM*, 36(12):36–53, 1993.

[8] Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder. Grapevine: An Exercise in Distributed Computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[9] Anita Borg, Wolfgang Blau, Wolfgang Graetsch, Ferdinand Heermann, and Wolfgang Oberle. Fault Tolerance Under UNIX. *ACM Transactions on Computer Systems*, 7(1):1–23, February 1989.

[10] T. Bressoud and F. Schneider. Hypervisor-based Fault Tolerance. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

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

[12] Michael Burrows, Charles Jerian, Butler Lampson, and Timothy Mann. On-line Data Compression in a Log-Structured File System. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 2–9, October 1992.

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

[14] J. Casas, D. Clark, R. Konuru, S. Otto, R. Prouty, and J. Walpole. MPVM: A Migration Transparent Version of PVM. In *Computing Systems*, volume 8, pages 171–216, Spring 1995.

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

[16] 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.

[17] David Cheriton and Dale Skeen. Understanding the Limits of Causally and Totally Ordered Communication. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 44–57, December 1995.

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

[19] Daniel Dias, William Kish, Rajat Mukherjee, and Renu Tewari. A Scalable and Highly Available Web Server. In *Proceedings of COMPCON*, March 1996.

[20] Whitfield Diffie and Martín Hellman. New Directons in Cryptography. In *IEEE Transactions on Information Theory*, pages 74–84, June 1977.

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

[22] Fred Douglis and John Ousterhout. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software - Practice and Experience*, 21(8):757–85, August 1991.

[23] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *IEEE/ACM Transacions on Networking*, November 1995.

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

[25] Sandy Fraser. Future Prospects for Wide Area Telecommunications. In *IEEE Micro*, February 1996.

[26] Alan Freier, Philip Karlton, and Paul Kocher. *Secure Socket Layer*. Netscape, March 1996.

[27] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The Digital Distributed System Security Architecture. In *Proceedings of the 12th National Computer Security Conference*, pages 305–319, 1989.

[28] D. Gelernter and D. Kaminsky. Supercomputing Out of Recycled Garbage: Preliminary Experience with Pirhana. In *Proceedings of Supercomputing '92*, pages 417–427, July 1992.

[29] Ian Goldberg, David Wagner, Randy Thomas, and Eric Brewer. A Secure Environment for Untrusted Helper Applications. In *Proceedings of the Sixth USENIX Security Symposium*, July 1996.

[30] C. Gray and D. Cheriton. Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 202–210, 1989.

[31] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[32] 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.

[33] James Gwertzman and Margo Seltzer. World-Wide Web Cache Consistency. In *Proceedings of the 1996 USENIX Technical Conference*, pages 141–151, January 1996.

[34] R. Haskin, Y. Malachi, W. Sawdon, and G. Chan. Recovery Management in Quicksilver. *ACM Transactions on Computer Systems*, 6(1):82–108, February 1988.

[35] 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.

[36] JavaSoft. *Java RMI Specification, Revision 1.1*, 1996. See http://chatsubo.javasoft.com/current/doc/-rmi-spec/rmiTOC.doc.html.

[37] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-Grained Mobility in the Emerald System. *ACM Transactions on Computer Systems*, 6(1):109–133, February 1988.

[38] Eric Dean Katz, Michelle Butler, and Robert McGrath. A Scalable HTTP Server: The NCSA Prototype. In *First International Conference on the World-Wide Web*, April 1994.

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

[40] R. Ladin, B. Liskov, L. Shirira, and S. Ghemawat. Providing Availability Using Lazy Replication. *ACM Transactions on Computer Systems*, 10(4):360–391, 1992.

[41] L.B. Mummert, M.R. Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain Resort, CO, December 1995.

[42] Matt M. Mutka and Miron Livny. The Available Capacity of a Privately Owned Workstation Environment. *Performance Evaluation*, 12(4):269–84, July 1991.

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

[44] Vern Paxon. End-To-End Routing Behavior in the Internet. In *Proceedings of the ACM SIGCOMM '96 Conference on Communications Architectures and Protocols*, August 1996.

[45] 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*, pages 169–177, December 1981.

[46] 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.

[47] M. Satyanarayanan, H. Mashburn, P. Kumar, D. Steere, and J. Kistler. Lightweight Recoverable Virtual Memory. *ACM Transactions on Computer Systems*, 12(1):33–58, February 1994.

[48] D. Scales and M. Lam. Transparent Fault Tolerance for Parallel Applications on Networks of Workstaions. In *Proceedings of the 1996 USENIX Summer Conference*, January 1996.

[49] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schi ller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the 1988 USENIX Conference*, March 1988.

[50] U.S. Geological Survey. Latest Quake Information. http://quake.usgs.gov/QUAKES/CURRENT/-current.html, May 1996.

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

[52] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl 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, December 1995.

[53] Marvin Theimer, K. Landtz, and David Cheriton. Preemptable Remote Execution Facilities for the V System. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pages 2–12, December 1985.

[54] Amin Vahdat, Paul Eastham, and Thomas Anderson. WebFS: A Global Cache Coherent File System. See http://www.cs.berkeley.edu/ vahdat/www6/-www6.html, 1996.

[55] Edward Wobber, Martin Abadi, Michael Burrows, and Butler Lampson. Authentication in the Taos Operating System. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pages 256–269, December 1993.

[56] Chad Yoshikawa, Brent Chun, Paul Eastham, Amin Vahdat, Thomas Anderson, and David Culler. Using Smart Clients to Build Scalable Services. In *Proceedings of the USENIX Technical Conference*, January 1997.

[57] Sognian Zhou, Jingwen Wang, Xiaohn Zheng, and Pierre Delisle. Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computing Systems. Technical Report CSRI-257, University of Toronto, 1992.