Operating Systems for the WWW

Robert Boyer Davis Chen
Abigail Gray Tim Lee
University of California at San Diego

Abstract. The evolution of inexpensive, powerful personal computers, ever-expanding storage capacity, gigabit bandwidth backbones, unique hardware devices and direct portal access has opened a new frontier in distributed computing. Several operating systems have been built in an attempt to harness the vast store of computational power locked in idle personal computers. A distributed operating system (OS) that seamlessly integrates a wide variety of devices into a computational grid is required. Computational abilities and overall performance should improve while users are guaranteed that their individual performance will not be reduced significantly and their privacy will be maintained

Any distributed OS hoping to meet these demands must support the concept of a *global system-wide OS, dynamic resource allocation, a file system which hides the physical location from the user, fault tolerance, protection,* and *scalability,* while simultaneously improving productivity. We describe each of these components and then evaluate ***Amoeba, Mach, NOW***, and ***WebOS*** against these core requirements. And finally, we review the feasibility of such an OS and offer some ideas for a distributed OS acceptable to users.

1 Introduction

The performance to cost ratio of personal computers and workstations continues to dramatically outpace that of mainframes and MPPs. [1] *Smaller* computers have multiplied and invaded our homes and offices. Concurrent with this trend is the availability of personal direct connections (cable modem, ADSL, and ISDN) and work force direct connections (T1 and above) to the rapidly growing World Wide Web. Also, rare hardware devices, such as electron microscopes, telescopes, and others, have been connected to the net.

The resulting web environment is different from most networks supporting distributed computing. It is a geographically distributed, heterogeneous computing environment with a high-speed communications backbone yet only moderate speed connectivity.

Users do not utilize their computers 24 hours a day; there is tremendous, untapped computational power available. Even by conservative estimates, personal computer and workstation resources are idle 70% of the time.

> "While the World Wide Web has made geographically distributed read-only data easy to use, geographically distributed computing resources remain difficult to access. As a result, wide-area applications that require access to remote CPU cycles, memory, or disk must be programmed in an ad-hoc and application-specific manner." [10]

Dozens of attempts have been made to tap the idle CPUs, DRAM, and disks. The difficulty involved is underscored by the marginal success of the best attempts at a distributed OS. Among the myriad complications, user concerns of decreased performance and security top the list. The current computational grid is merely awaiting an OS to take advantage of its power in a way that is transparent to the common user.

In this paper we present and extend Tanenbaum's requirements for a distributed OS. [6] Then we introduce each of the systems surveyed, Amoeba, Mach, NOW, and WebOS and compare them against our distributed OS model for the web. Finally, we analyze the successes and reevaluate the distributed OS definition for practicality.

2 Distributed OS Requirements

> "A *distributed* [OS] is one that looks to its users like an ordinary centralized [OS] but runs on multiple, independent CPUs. The key concept here is *transparency*." [6]

The following are required to have a truly distributed OS.

2.1 Global, System Wide OS

A single conceptual OS must exist such that when a hardware device boots it is added as an integral part of the system. This contrasts with most machines connected to the web that each have their own OS and maintain a substantial firewall to the web.

A global, system wide OS merges the concepts of a network OS and a computational grid. It provides an integrated environment for all hardware and users.

The grid should be able to incorporate devices at any time, making its resources available for all to use based on access rights. With the vast quantities of computers and devices connected to the computational grid, the ability to find resources is crucial.

Managing access to resources is a complex issue involving a combination of protection mechanisms and kernel level micro-cash accounting and possibly a billing service to support.

Users login to the system using a globally unique identifier. This allows a user to login from anywhere in the world. It should also be used to associate the user with their data files instead of physical location.

2.2 Dynamically Allocate Resources

Dynamically allocating resources frees the user from the limitations of their machine in terms of CPU, memory, and secondary storage. In order for dynamic resource allocation to be practical & scalable, high-bandwidth, low latency networks with low overhead network interfaces are required. The speed of the network allows resources to be viewed as a shared pool, thereby expanding the universe of available resources.

2.2.1 CPU Allocation

A user should be able to work on any work-station in the network without having to perform some form of *remote* login.

This takes on three distinct behaviors, one of dynamically allocating an entire application but for the display (if any), dynamically allocating subroutines in a kind of read-ahead cache, and dynamically allocating threads for parallel processes.

Communicating processes cannot rely on location dependent schemes such as IP addresses, but must rely on the level of virtualization provided by the system. Further, if processes are allowed to dynamically migrate, open communications channels must also move.

2.2.2 Memory Allocation

In a fully efficient system, available idle memory will be accessible to processes in need as a form of virtual memory to enhance performance.

2.3 File System

The OS should handle file placement management and the physical location of files should be hidden from the user. Typically, users are aware of where each of their files are kept and must move files between machines with explicit *file transfer* commands instead of having file placement managed by the OS. [6]

Critical aspects of a distributed operating system's transparent file system are replication, multi-user consistency, disconnected operations, privacy and accessibility

2.4 Fault Tolerance

Users should be shielded from the effects of hardware failure. Tanenbaum [6] states,

> "If 1 percent of the personal computers crash, 1 percent of the users are out of business, instead of everyone simply being able to continue normal work albeit with 1 percent worse performance."

Although he does not address the fact that those one- percent may no longer have a keyboard, monitor, or I/O controller available, his point is well intentioned.

2.5 Protection

Protection is distinctly at odds with global computing efficiency. Protection is required for

guarding private data: on the wire, on remote unsecured disk and in remote memory, for *guarding remote programs*: on the wire and during remote execution, and for *guarding hardware* against unauthorized use and attempts to disable service.

Mechanisms such as secure cache-coherent global file systems, authentication for fine-grained capability-based access control, and secure execution of remote programs begin to address these issues.

2.6 Scalability

Scalability describes how well an OS design incorporates additional users, hardware and other resources. In order to scale well, an operating system should transparently handle the complexity of additional resources, users and network traffic.

3 Operating Systems Reviewed

The intent and assumptions made by each the operating systems reviewed follows.

3.1 Amoeba

The primary goal of the Amoeba project was to build a globally transparent distributed OS. This means that there is no concept of a local or home machine, a user logs into the system without any awareness of a particular physical machine. Similarly, programs are run by the system as a whole and not by a particular machine. A user views and accesses all system wide resources as a whole and has the privilege exploiting all system wide processing power.

The architecture of Amoeba is based on the processor pool model. It incorporates a vast number of CPUs into a globally uniform system. The architecture consists of 4 primary elements, *workstations*, *processor pools*, *specialized servers*, and *gateways*. Workstations are the interface to the system. The processor pool is the collection of all CPUs in the system. The processing power is dynamically allocated to each user to balance loads on the system. Specialized servers, such as file or directory servers, provide system users with a particular service or function. The gateways interface with wide-area networks to link Amoeba systems together in a uniform and transparent manner.

To enhance reliability and promote flexibility, Amoeba acts as a micro kernel implemented in user space. Amoeba uses a client and server model of communication based on remote procedure calls and return service results. Fault detection,retransmission, and acknowledgement are provided to ensure reliable message passing. Amoeba specifically avoids using shared memory for scalability reasons.

3.2 Mach evolved from the Rochester Intelligent Gateway and Accent. Accent focused on combining memory mapping with message passing. Mach extends Accent from a network OS for an uniprocessor to a new computing environment for multiprocesssors connected on high-speed network. There are four basic abstractions that Mach inherited from Accent, *task/process*, *thread*, *port*, and *message*. Their purpose is to provide control over execution, virtual memory management and inter-process communication.

Mach also provides the concept of a memory object, which most sets it apart from Accent. These give Mach the ability to efficiently manage system services such as network paging and file system support outside the kernel, in what we today call an exokernel. Mach provides a small set of basic facilities, which provides great flexibility for implementing system software and OS applications.

3.3 Network Of Workstations (NOW)

Implemented at UC Berkeley, NOW's main component is GLUnix, an OS layer placed over a cluster of workstations each running UNIX. NOW provides transparent remote execution, support for interactive parallel and sequential jobs and load balancing.

NOW improves virtual memory and file system performance by using the aggregate DRAM of individual machines as a giant cache. This achieves cheap, highly available, scalable file storage by using redundant arrays of work-

station disks. The LAN functions as the I/O backbone. NOW's design requires efficient communication between hardware and software, global coordination of multiple workstation OS's and enterprise-scale network file systems.

3.4 WebOS

WebOS took the design approach of enabling new technology without making serious modifications to the existing OS. It is symbiotic with Unix workstations and works within the environment of the World Wide Web to provide general purpose distributed computing over the Web. It is an OS designed specifically to provide the services needed to build geographically distributed, highly available, incrementally scalable, and dynamically reconfiguring applications. It integrates simple protocols from existing distributed systems into the Web to enable a two-way computing on the web - a whole new class of distributed applications. [10]

> "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)." [10]

WebOS demonstrates the power of providing a common set of OS services to wide-area applications, including mechanisms for naming, persistent storage, remote process execution, resource management, authentication, and security. The claim is substantiated by their Rent-A-Server used to enable overly burdened servers to dynamically shed load during peak access times. New servers are dynamically spawned and requests redirected to take advantage of locality.

4 Global, System Wide OS

The OS's reviewed ranged from the centralized control of Amoeba to the independent OS-per-node requirement of NOW. Mach takes a minimalist approach providing more of a global network than an OS.

4.1 Amoeba is a globally transparent distributed operating system in which the system appears as a single computer to the user. The user is unaware of the existence of any remote machine. There is no concept of a local or home machine, a user logs into "the system" without any distinction of a particular physical machine. Similarly, works are being done by the system as a whole and not by any particular machine. Computing powers are dynamically allocated by the system to balance processing load across the entire Amoeba system. Thus, all system wide resource is view as a whole in Amoeba.

4.2 Mach has no sense of a global system, the designed approach uses a workstation model with users specifically login to a home station. Load sharing is done over the collection of CPUs on the home machine and not over the collection of machines each possibly with multiple CPUs. Similarly there is no sense of global resources, access to remote resource must be done through explicit remote access.

4.3 NOW is not a global OS in that it requires each member computer to run its own fully functional UNIX OS. However, NOW does provide several OS functions at a global level: resource allocation, load balancing and processor scheduling to name a few.

4.4 WebOS is not a global OS. Rather, it takes the approach of a small component of a library operating system for an exokernel. It focuses on the services provided and does not discuss the user. It provides for a global naming but this occurs on the level of processes for the purpose of CPU loading and the file system for data access**.**

5 Dynamic Resource Allocation

The raison d'être for global system wide operating systems; each system reviewed provides some level of dynamic resource allocation. NOW distributes load only on job startup while WebOS supports process migration. NOW has broader definition of *resource* than the other operating systems, in addition to sharing CPU time, NOW shares DRAM and disk access as well. To spread such behavior across the web will require secure, trusted, micro-cash cost accounting built directly into the kernel with mechanisms to query usage costs before borrowing resources.

5.1 Amoeba has a central processor pool for parallel and distributed computations. Each process has its own stack and its own program counter, so that when one of the processes blocks on a remote procedure call, the others will not be affected. A process server who keeps track of the availability of processors controls the processor pool. Services in Amoeba are separated to provide the service independently from each other. Such separation enhances the overall system performance and fault tolerance.

5.2 Mach has the concept of a processor set which has a collection of CPUs at its disposal and a collection of threads that need computing power. Each processor set is a closed world, with its own resources and its own clients, independent of all the other processor sets. A process may dynamically assigns threads to a processor set as work proceeds, keeping the load balanced only within the home machine**.**

5.3 NOW provides load balancing on job startup through intelligent placement of processes. NOW does not provide job migration, which limits NOW's ability to utilize idle resources on personal computers. The GLUnix master layer keeps track of available resources. Each computer node runs a daemon that periodically queries the local job load and reports to the master. The daemon uses two functions to keep track of the current load: glups queries all current running jobs and glustat queries the current status of all NOW machines. To begin a new job, an application calls GLURun with the parallel degree of the program and user's current environment (group, cwd, umask, and environment variables). Using stored information, the GLUnix master finds the node with lowest processor load and sends the job to that node to be run. Binary programs can be copied and executed on more than one node at a time to improve performance.

5.4 WebOS claims dynamic process allocation is as simple as forking a process on the local processor. They achieve this by providing a resource manager on each machine responsible for job requests from remote sites. To maintain local system integrity, a virtual machine is created which interfaces with the security system to enforce rights.

Ideally users employ a single name for the Web service and the system translates the name to the IP address of the replica that will provide the best service to the client. WebOS approaches this ideal by loading application and server specific code into end clients to perform name translation. These *Smart Clients* enable extensions of the server functionality to be dynamically loaded onto the client machine (using Java applets).

> "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 the best service to the user. The decision is made in a service-specific manner." [10]

The customizable graphical interface thread implements the user's view of the service. The director thread maintains the necessary state to perform fail over if a service provider becomes unavailable, thus transparently masks individual failures.

6 File System Hides Physical Location

Tanenbaum presents transparent file location as a requirement for a distributed OS. Freeing the user from physical file locations is critical to users being able to login anywhere on a system and being able to work as if they were on their home node.

6.1 Amoeba does not dictate the choice of a specific file system. The file system is simply a collection of server processes in user space. The micro kernel is independent from the file system. In addition, Amoeba supports the simultaneous use of different and even incompatible file systems**.**

Amoeba threads within the same process share address space, however, each thread has its own registers, program counter, and stack. The Amoeba file server utilizes multi threading to independently handle incoming requests. By splitting the server up into multiple threads, each thread can be purely sequential, even if it has to block waiting for I/O. Nevertheless, all the threads can have access to a single shared software cache. The kernel uses semaphores and mutexes to prevent threads from accessing the shared cache simultaneously.

A file server uses the Object Field in the capability to identify the file of client's interest. The Object Field in the capability of Amoeba file server is analogous to UNIX's i-node number.

Amoeba's file server file system indeed hides the physical location of the file from the users by when performing an operation on an object, it is not necessary to know where the object resides.

6.2 Mach uses paging to support shared virtual address space. A key concept relating to the use of virtual address space is the idea of a memory object. A memory object can be a page or a set of pages, but it can also be a file or other, more specialized data structure. A memory object, which may be a file may be read and written once it is mapped into the virtual address space. The read and write system calls allow a thread to access virtual memory belonging to another process. These calls require the caller to have possession of the process port belonging to the remote process, which may be granted by the remote process. Shared memory plays an important role in Mach. All threads in a process see the same address space and have access to the data pages or files. A memory manager that runs in the user space controls the mapping of memory objects into process' address space. Since communication is transparent in Mach, a memory manager need not reside on the machine whose memory it is managing. A Distributed Shared Memory server provides access to memory objects by mapping the memory object to machine address space. This approach blurs the awareness of file's physical location from the user.

6.3 NOW Because each computer runs its own local OS, a full global file system is not needed. However jobs executed remotely should have the illusion they are being run locally. To begin a job, an application calls Glurun and passes in the environment the job will run it. Jobs executed remotely will produce the same effect as the same job run remotely. There are a couple exceptions to this where the job location is not transparent. The authors consider these errors and offer suggestions for overcoming them.

6.4 WebOS chose URL[1]s as their global naming scheme to maintain consistency with existing systems. They successfully achieved their design goal to make the system work with existing applications without kernel modifications.

The costs associated with coherency or authentication should be incurred only by accesses requiring these services. Because capabilities did not exist, WebFS was built. To provide global file service for unmodified

[1] URL - Universal Resource Locator

applications, WebFS intercepts all file system related system calls using SLIC.[2] It allows for interoperability with existing applications without any kernel source modifications. To mitigate the context switch overhead of forwarded system calls to the user-level, SLIC allows servers to register filters which forward only system calls with particular arguments. In the case of WebFS, only file system calls requesting URLs need to be sent to the user-level server.

WebFS provides a secure cache coherent global file system, however its policy is *last writer wins*. WebFS maintains a table of open remote file descriptors and a disk cache of remote file pages. When the WebFS on a machine detects a page has been modified, it sends an invalidate message to all other nodes that have cached the page. Programs to open, read, stat, access, close were implemented with CGI to provide required file system functionality while not disturbing the existing OS.

7 Fault Tolerance

In the systems we reviewed, the extent of fault tolerance is in direct proportion to the size of the user base, keeping in mind all are used for research purposes. In a production environment fault tolerance would be given greater priority.

7.1 Amoeba's fault tolerance design is based on the idea that server crashes are infrequent and thus it is not necessary to make crashes completely transparent to the users

Amoeba provides a boot service, which other servers may register, the boot server periodically polls each registered server to determine its operational state. If the boot server times out waiting for response from a particular server, the boot server will declare that particular server to be down and request the processor pool to start up a new copy of that server.

[2] *SLIC is a loadable device driver that patches the Unix OS syscall table to allow forwarding of system calls to user-level servers.*

The concept of a virtual circuit or a session is not part of Amoeba's communication philosophy. Therefore each remote procedure call transaction is self contained and independent of prior setup. If a server crashes during a transaction, the client simply timeouts out and try again. The only exception to the problem is when the transactional operation is not idempotent, however this is rarely the case. Amoeba sacrifices this for the benefit of low overhead.

Due to the centralized design of the system, the crux of the system lies with the primary server. If it goes down, so does the entire network.

7.2 Mach is very flexible in fault tolerance issues because it all depends on the particular OS application emulation or user space server implementation. Recall that one of the primary design goals of Mach is to facilitate the development of efficient user space software. The degree of fault tolerance is left as a policy for the emulated OS application or the servers that run in the user space**.**

7.3 NOW is able to tolerate the failure of any number of nodes, but not the failure of the node running the GLUnix master. The master keeps an open connection to all running nodes as well as a list of which processes are running on each node. If a connection to a node is severed, the master looks up which processes were running on that node and marks them as killed. The start-up job or parent process is notified of the death. NOW does not provide checkpoints or automatic restarts. In fact, every user has the ability to perform a manual restart of the entire system.

7.4 WebOS addresses fault tolerance by having its Smart Clients maintain enough state that fail over is possible. It also manages the fault of overloaded hardware for which it sheds load – dynamically by spawning another server and redirecting requests. WebOS took a transaction

approach, that applications must have well-defined failure modes. For example, an aborted remote agent should not leave a user's file system in an inconsistent state

8 Protection

Protection concerns are perhaps the largest hurdle facing global OS hopefuls. Any "global" system will include malicious users; the larger the system, the greater the risk. The systems reviewed walk a fine line between cooperation and concealment. NOW restricts its user base in a way that is not feasible for an operational system.

8.1 Amoeba Resources are objects, objects have associated capabilities, thus protection is provided**.**

Amoeba is based on the conceptual model of abstract data types or objects which is managed by servers providing the service. Amoeba provides basic protection mechanism but support a more complicated security policy depending on the emulation application executed in the user space.

All naming and protection issues in Amoeba are dealt with by a single uniform mechanism called the sparse capabilities. System resources, including file, directories, and disks are treated as object and each object is owned and managed by a particular server which provides the service. For example, the file server will own a file object.

Each object has a globally unique name and associated capabilities. Capabilities are managed entirely by user processes without the involvement from kernel.

Capability has a check field. It is an encryption parameter generated to authenticate the capability for protection purposes.

8.2 Mach – Inter-process communication in Mach is based on message passing. Messages are send or received from a process port. Permission to send or receive from a process port takes the form of a capability. Process port is used to communicate with the kernel. Each thread also has its own thread port, which is used to invoke its private kernel services. Threads are managed by kernel, thus they are sometimes called heavyweight threads rather than the traditional lightweight thread which is pure user space threads.

A Mach process does not have a uid, gid, signal mask, root directory, working directory, or file descriptor array, all of which UNIX processes do have. All of this information is managed by the emulation package, so the Mach micro kernel knows nothing at all about it.

8.3 NOW –was created to run in a trusted network on top of complete operating systems. Once a user has logged into a local machine, he is implicitly trusted. NOW was implemented with a maximum of 100 nodes in a research environment. Protection will be major consideration if NOW's scope of use is expanded. For practical purposes, a user can reserve a node for a set time. At this time only the node "owner" can run processes on that node.

8.4 WebOS – provides a secure cache coherent global file system, authentication for fine-grained capability-based access control, and secure execution of remote programs.

WebOS goals were to provide a simple mechanism for reasoning about granting access to foreign programs; provide users with a way to transfer access rights to programs running remotely on their behalf, and to not incur over-head associated with authentication when accessing public resources.

WebOS chose public key cryptography and assume the existence of trusted certification authorities to guarantee the association of public keys to entities.

The WebFS server maintains an access control list (ACL) for read, write, and execute access to all local files. One of the arguments to system calls such as *open* is an identity

certificate. The certificate is also used to guard against replay attacks.

Protection against the risks of remote execution is provided by constraining processes to run on a restricted virtual machine isolated from other programs, in a *sandbox.* Such programs may be allotted limited resources, may not be allowed to access most of the local file system, and may be forbidden from making arbitrary network connections. To manage their sandbox, WebOS chose secure remote helper application (SRHA) techniques to enable the system to interact with unmodified applications. SRHA uses OS tracing facilities to intercept and disallow the subset of system calls that could possibly violate system integrity. That work was extended to provide sandboxing of arbitrary executables using SLIC to intercept the needed system calls.

9 Scalability

The systems reviewed all scale well within certain limiting assumptions or design decisions. For example, both GLUnix of NOW and WebOS make specific use of Unix features not available in other operating systems.

9.1 Amoeba . Targeting to incorporate a large number CPUs in a straightforward way.

In order to scale the system, Amoeba avoids the idea of CPUs sharing physical memory. If shared memory is present, it can be utilized to optimize message passing by just doing memory-to-memory copying instead of sending messages over the LAN.

Amoeba avoids using shared memory for scalability reason. While it would be easy to build a 16 node shared memory multiprocessor, it would not be easy to build a 1000 node shared memory multiprocessor.

Originally Amoeba is primarily used on local area networks. Recently, there has been some work developed on the wide area networks. What LAN differs from WAN is that WAN lacks the broadcasting; thus, Amoeba takes a different approach on WAN. Amoeba's approach to WAN requires the owner, which is the human user, of the service to explicitly publish the service port to the gateways. Then when the client kernel broadcasts the LOCATE packet, the gateway kernel responds in the usual way. Only the gateway knows that WAN communication is involved, the RPC is transparent to the client and the server .

9.2 Mach – The Mach distributed OS support an implementation a distributed shared memory server [3], which promotes scalability. The distributed solution allocated multiple memory servers to avoid the bottleneck of excess paging request as the system grows in size.

9.3 NOW – The NOW design requires each node to run a version of the Unix workstation. There is another version of GLUnix that runs with LINIX. The authors are currently working on a tool that would allow NOW to be used with commercially marketed OS's. Since NOW has a centralized master it can be used with only limited number of nodes. With a limited number of nodes (less than 100) the centralized master is not a performance bottleneck. Only 1% of job time is spend in the master. Network and file system interactions are more costly than services provided by the master. To increase the node limit, NOW would need to be redesigned using a peer to peer protocol.

9.4 WebOS – is well suited to questions of scalability. It is built on top of the web infrastructure as a framework to support general computation on the web using a common set of protocols. Issues might be cache coherent file system; performance and bandwidth costs associated with authentication; ability to access files WebFS**.**

10 Conclusion

None of the operating systems reviewed quite measure up to the definitions of a distributed OS set forth by Tanenbaum and applied to the web environment.

His own operating system, Amoeba, follows the definitions most closely, but is not sufficient for the diverse web environment.

Although Mach does not support the concept of a global system wide OS and its dynamical resource allocation schemes are weak, it does have a form of exokernel to provide a small set of basic facilities outside the kernel.

NOW provides a unifying wrapper around Unix through the use of GLUnix. With this, users are able to capitalize on the untapped power of idle resources – CPU, memory, disk. NOW works well in a LAN environment due to the availability of high speed, low latency bandwidth. However, today there is probably not enough bandwidth to support it on the web.

WebOS uses the Unix SLIC to avoid the cost of kernel crossings and provides a set of OS services to improve resource utilization across the web

In review, we find Tanenbaum's ideas are sound, but his writings are eclipsed by the rapid changes which have brought us the World Wide Web. Given human nature, the concept of a global, system wide OS is both unachievable and technically undesirable. Unique hardware and performance considerations will force us to maintain unique OS per device type. These systems seem to have the right idea of removing layers of access to kernel functions. Exokernel-like techniques, similar to those developed for Mach, NOW and WebOS, are a step in the right direction.

Challenges to a widely used, distributed OS include managing unique user names, managing file concurrency, replication, and accessibility, fault tolerance, heterogeneous computing, transactional processing, security infrastructure providing secure transactions and private data, and scalability.

Further, to develop a fully collaborative confederation of computing resources, the web equivalent to the global system-wide OS, a common micro-cash accounting system and standard protocols which all participating operating systems can incorporate are required.

11 Bibliography

1. Anderson, T., Culler D, Patterson, D. and the NOW team."A Case for Now (Networks of Workstations)", Dec 1994, src unknown.
2. Denning, P.J., Browne, J.C., Peterson, J.L., "The Impact of Operating Systems Research on Software Technology," *Research Directions in Software Technology*, MIT Press, Cambridge, Mass, (June 1979), pages 490-513.
3. Forin, A., Barrera, J., Young, M., Rashid, R., “Design, Implementation and Performance Evaluation of a Distributed Shared Memory Server for Mach, *USENIX Conference*, January 1989.
4. Ghormley, D., Petrou, D, Rodrigues, S., Vahdat, A., Anderson, T., “GLUnix: a Global Layer Unix for a Network of Workstations” University of CA at Berekely, August 14, 1997.
5. Rashid, R., Baron, R., Forin, A., Golub, D., Jones, M., Julin, D., Orr, D., Sanzi, R., “Mach: A System Software Kernel,” *Computer Society International Conference* COMPCON 89, February 1989.
6. Tannebaum, A.S., Renesse, R. van, "Distributed Operating Systems," 1986
7. Tannebaum, A.S., Kaashoek, M.F., Renesse, R. van and Bal, H., "The Amoeba Distributed Operating System – A Status Report," *Computer Communications* vol. 14, pp. 324-335, July/Aug. 1991
8. Tannebaum, A.S., Distributed Operating Systems, Prentice-Hall, 1995. p. 376~472
9. Vahdat, A., Dahlin, M., Anderson, T., "Turning the Web Into a Computer," University of California at Berkeley.
10. Vahdat, Amin, Ed. "WebOS: Operating System Services for Wide Area Applications," http://now.cs.berkeley.edu/WebOS/index.html, Oct 99.
11. Young, M., et al, “The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System,” Carnegie Mellon University, Nov., 1987.