The Amoeba Distributed Operating System

Andrew S. Tanenbaum
Vrije Universiteit
De Boelelaan 1081a
Amsterdam, The Netherlands
FAX: +31 20 6427705
Email: ast@cs.vu.nl

1. INTRODUCTION

Roughly speaking, we can divide the history of modern computing into the following eras:

- 1970s: Timesharing (1 computer with many users)
- 1980s: Personal computing (1 computer per user)
- 1990s: Parallel computing (many computers per user)

Until about 1980, computers were huge, expensive, and located in computer centers. Most organizations had a single large machine.

In the 1980s, prices came down to the point where each user could have his or her own personal computer or workstation. These machines were often networked together, so that users could do remote logins on other people's computers or share files in various (often ad hoc) ways.

Nowadays some systems have many processors per user, either in the form of a parallel computer or a large collection of CPUs shared by a small user community. Such systems are usually called *parallel* or *distributed computer systems*.

This development raises the question of what kind of software will be needed for these new systems. To answer this question, a group under the direction of Prof. Andrew S. Tanenbaum at the *Vrije Universiteit* (VU) in Amsterdam (The Netherlands) has been doing research since 1980 in the area of distributed computer systems. This research, partly done in cooperation with the *Centrum voor Wiskunde en Informatica* (CWI), has resulted in the development of a new distributed operating system, called *Amoeba*, designed for an environment consisting of a large number of computers.

When Amoeba had reached the point where it was beyond the laboratory stage and was ready for commercial use, VU and CWI decided to join forces with ACE, a leading Dutch R&D company. While the research people will continue to push the software technology forward, ACE will handle the marketing and most importantly, provide commercial support of Amoeba to customers requiring it.

Amoeba is available for free to universities and other educational institutions and for standard commercial prices and conditions to corporate, government, and other users, as described later.

2. WHAT IS AMOEBA?

Amoeba is a general-purpose distributed operating system. It is designed to take a collection of machines and make them act together as a single integrated system. In general, users are not aware of the number and location of the processors that run their commands, nor of the number and location of the file servers that store their files. To the casual user, an Amoeba system looks like a single old-fashioned time-sharing system.

Amoeba is an ongoing research project. It should be thought of as a platform for doing research and development in distributed and parallel systems, languages, protocols and applications. Although it provides some UNIX emulation, and has a definite UNIX-like flavor (including over 100 UNIX-like utilities), it is NOT a plug-compatible replacement for UNIX. It should be of interest to educators and researchers who want the source code of a distributed operating system to inspect and tinker with, as well as to those who need a base to run distributed and parallel applications.

Amoeba is intended for both "distributed" computing (multiple independent users working on different projects) and "parallel" computing (e.g., one user using 50 CPUs to play chess in parallel). Amoeba provides the necessary mechanism for doing both distributed and parallel applications, but the policy is entirely determined by user-level programs. For example, both a traditional (i.e. sequential) ’make’ and a new parallel ’amake’ are supplied.

3. DESIGN GOALS

The basic design goals of Amoeba are:

- Distribution—Connecting together many machines
- Parallelism—Allowing individual jobs to use multiple CPUs easily
- Transparency—Having the collection of computers act like a single system
- Performance—Achieving all of the above in an efficient manner

Amoeba is a *distributed* system, in which multiple machines can be connected together. These machines need not all be of the same kind. The machines can be spread around a building on a LAN, and multiple LANs can be connected together over wide-area networks (WANs), such as the Internet.

Amoeba is also a *parallel* system. This means that a single job or program can use multiple processors to gain speed. For example, a branch and bound problem such as the Traveling Salesman Problem can use tens or even hundreds of CPUs, if available, all working together to solve the problem more quickly. Large ‘‘back end’’ multiprocessors, for example, can be harnessed this way as big ‘‘compute engines.’’

Another key goal is *transparency*. The user need not know the number or the location of the CPUs, nor the place where the files are stored. Similarly, issues like file replication should be handled largely automatically, without manual intervention by the users.

Put in different terms, a user does not log into a specific machine, but into the system as a whole. There is no concept of a ‘‘home machine.’’ Once logged in, the user

does not have to give special *remote login* commands to take advantage of multiple processors or do special *remote mount* operations to access distant files. To the user, the whole system looks like a single conventional timesharing system.

Performance and reliability are always key issues in operating systems, so substantial effort has gone into dealing with them. In particular, the basic communication mechanism has been optimized to allow messages to be sent and replies received with a minimum of delay, and to allow large blocks of data to be shipped from machine to machine at large bandwidth. These building blocks serve as the basis for implementing high performance subsystems and applications on Amoeba.

4. SYSTEM ARCHITECTURE

Since distributed and parallel computing is different from personal computing, it is worth while first describing the kind of hardware configuration for which Amoeba was designed. A typical Amoeba system will consist of four functional classes of machines. First, each user has a workstation for running the user interface, the X window system. This workstation can be a typical engineering workstation, or a specialized X terminal. It is entirely dedicated to running the user interface, and does not have to do other computing.

Second, there exists a pool of processors that are dynamically allocated to users as required. These processors can be part of a multiprocessor or multicomputer, be a collection of single-board computers or be a group of workstations allocated for this purpose. Usually, each pool processor has several megabytes of private memory, that is, pool processors need not have any shared memory (but it is not forbidden). Communication is performed by sending packets over the LAN. All the heavy computing happens in the processor pool.

Third, there are specialized servers, such as file servers and directory servers that run all the time. They may run on processor pool processors, or on dedicated hardware, as desired.

Finally, there are gateway machines that allow multiple Amoeba systems that are far apart to be connected together in such a way that to the user, the whole appears to be a single integrated system, rather than a collection of different systems.

All these components must be connected by a fast LAN. At present only Ethernet is supported, but ports to other LANs are possible.

5. THE AMOEBA MICROKERNEL

The following sections briefly provide an introduction to the Amoeba microkernel and some of its characteristics.

5.1. Microkernel + Server Architecture

Amoeba was designed with what is currently termed a *microkernel* architecture. This means that every machine in an Amoeba system runs a small, identical piece of software called the *kernel*. The kernel supports the basic process, communication, and object primitives. It also handles raw device I/O and memory management. Everything else is built on top of these fundamentals, usually by user space *server* processes.

Thus the system is structured as a collection of independent processes. Some of these are user processes, running application programs. Such processes are called *clients*. Others are *server* processes, such as the Bullet file server or the directory server. The basic function of the microkernel is to provide an environment in which clients and servers can run and communicate with one another.

This modular design makes it easier to understand, maintain, and modify the system. For example, since the file server is an isolated server, rather than being an integral part of the operating system, it is possible for users to implement new file servers for specialized purposes (e.g. NFS, database). In conventional systems, such as UNIX, adding additional user-defined file systems is infeasible.

5.2. Threads

In UNIX and many other traditional operating systems, a process consists of an address space and a single thread of control. In Amoeba, each process has its own address space, but it may contain multiple ‘‘threads of control’’ (*threads*). Each thread has its own program counter and its own stack, but shares code and global data with all the other threads in its process.

Having multiple threads inside each process is convenient for many purposes and fits into the model of distributed and parallel computing very well. For example, a file server may have multiple threads, each thread initially waiting for a request to come in. When a request comes in, it is accepted by some thread, which then begins processing it. If that thread subsequently blocks waiting for disk I/O, other threads can continue. Despite their independent control, however, all the threads can access a common block cache, using semaphores to provide inter-thread synchronization. This design makes programming servers and parallel applications much easier.

Not only are user processes structured as collections of threads communicating by RPC, but the kernel itself is as well. In particular, threads in the kernel provide access to memory management services.

5.3. Remote Procedure Call

Threads often need to communicate with one another. Threads within a single process can just communicate via the shared memory, but threads located in different processes need a different mechanism. The basic Amoeba communication mechanism is the *remote procedure call* (RPC). Communication consists of a client thread sending a message to a server thread, then blocking until the server thread sends back a return message, at which time the client is unblocked.

To shield the naive user from these details, special library procedures, called *stubs*, are provided to access remote services. Amoeba has a special language called *Amoeba Interface Language* (AIL) for automatically generating these stub procedures. They marshal parameters and hide the details of the communication from the users.

5.4. Group Communication

For many applications, one-to-many communication is needed, in which a single sender wants to send a message to multiple receivers. For example, a group of cooperating servers may need to do this when a data structure is updated. It is also frequently needed for parallel programming. Amoeba provides a basic facility for reliable, totally-ordered group communication, in which all receivers are guaranteed to get all group messages in exactly the same order. This mechanism simplifies many distributed and parallel programming problems.

5.5. Memory Management

The Amoeba memory model is simple and efficient. A process' address space consists of one or more segments mapped onto user specified virtual addresses. When a process is executing, all its segments are in memory. There is no swapping or paging at present, thus Amoeba can only run programs that fit in physical memory. The primary advantage of this scheme is simplicity and high performance. The primary disadvantage is that it is not possible to run programs larger than physical memory.

5.6. Input/Output

I/O is also handled by kernel threads. To read raw blocks from a disk, for example, a user process having the appropriate authorization, does RPCs with a disk I/O thread in the kernel. The caller is not aware that the server is actually a kernel thread, since the interface to kernel threads and user threads is identical. Generally speaking, only file servers and similar system-like processes communicate with kernel I/O threads.

6. SOFTWARE OUTSIDE THE KERNEL

The job of the Amoeba microkernel is to support threads, RPC, memory management and I/O. Everything else is built on top of these primitives.

6.1. Objects and Capabilities

Just as there are two unifying concepts in the kernel, threads and communication, there are also two unifying concepts in the user-level software: objects and capabilities. An *object* is conceptually an abstract data type. That is, an object is a data structure on which certain operations are defined. For example, a directory is an object to which certain operations can be applied, such as ''enter file'' and ''look up name.''

Amoeba primarily supports software objects, but hardware objects also exist. Each object is managed by a server process to which RPCs can be sent. Each RPC specifies the object to be used, the operation to be performed, and any parameters to be passed.

When an object is created, the server doing the creation constructs a 128-bit value called a *capability* and returns it to the caller. Subsequent operations on the object require the user to send its capability to the server to both specify the object and prove the user has permission to manipulate the object. Capabilities are protected cryptographically to prevent tampering. All objects in the entire system are named and protected using this one simple, transparent scheme.

6.2. Bullet File Server

The standard Amoeba file server has been designed for high performance and is called the *Bullet server*. It stores files contiguously on disk, and caches whole files contiguously in core. Except for very large files, when a user program needs a file, it will request that the Bullet server send it the entire file in a single RPC. A dedicated machine with at least 16 MB of RAM is needed for the Bullet file server for installation (except on the Sun 3 where there is a maximum of 12 MB). The more RAM the better, in fact. The performance is improved with a larger file cache and the maximum file size is limited by the amount of physical memory available to the Bullet server.

6.3. Directory Server

In contrast to most other operating systems file management and file naming are separated in Amoeba. The Bullet server just manages files, but does not handle naming. It just reads and writes files, specified by capabilities. A capability can be thought of as a kind of handle for an object, such as a file. A directory server maps ASCII strings onto capabilities. Directories contain (ASCII string, capability) pairs; these capabilities will be for files, directories, and other objects. Since directories may contain capabilities for other directories, hierarchical file systems can be easily built, as well as more general structures.

A directory entry may contain either a single capability or a set of capabilities, to allow a file name to map onto a set of replicated files. When the user looks up a name in a directory, the entire set of capabilities is returned, to provide high availability. These replicas may be on different file servers, potentially far apart (the directory server has no idea about what kind of objects it has capabilities for or where they are located). Operations are provided for managing replicated files in a consistent way.

6.4. Compilers

Amoeba comes standard with compilers for ANSI standard C, Pascal, Modula 2, BASIC, and Fortran 77. Each of these comes with appropriate libraries. Amoeba also comes with a collection of third-party software, including the GNU C compiler.

6.5. Parallel Programming

Amoeba also comes with a new language for parallel programming, Orca. Orca allows programmers to create user-defined data types and have processes on different machines to share them in a controlled way, in effect simulating an object-based distributed shared memory over a LAN. Operations on each object are performed in such a way as to provide the illusion of their being only a single copy, shared by all machines. The Orca run-time system uses the Amoeba group communication facility to make sharing of software objects over the network highly efficient. Orca is not yet available for the SPARC, but is supplied for the other supported architectures.

6.6. Utilities

Amoeba provides a large number of utilities modeled after the programs that come with UNIX. Among others, these include *awk, basename, cal, cat, cdiff, chmod, cmp, comm, compress, cp, cpdir, dd, diff, echo, ex, expr, factor, file, find, fold, fortune, grep, head, jove, kill, ksh, ln, look, ls, m4, make, mkdir, more, mv, od, pr, prep, printenv, pwd, quote, rev, rm, rmdir, sed, sh, shar, size, sleep, sort, spell, split, strings, sum, tail, tar, tee, termcap, test, time, touch, tr, treecmp, true, tset, tsort, tty, uniq, uud, uue, vi, wc, who, xargs, yacc* and many other old favorites. Furthermore, a number of new programs are provided such as *amake*, a highly parallel configuration manager.

6.7. UNIX Emulation

To aid in porting UNIX programs to the Amoeba environment, an emulation library, called Ajax, offers major POSIX P1003.1 compatibility. Most POSIX conformant programs work without modification. They simply have to be compiled and linked on Amoeba.

6.8. TCP/IP

Although the basic communication mechanism in Amoeba is the Amoeba FLIP protocol, a user-level server is provided to allow TCP/IP communication, through RPCs to the TCP/IP server. In this way, machines can be accessed through the Internet.

6.9. X Windows

Amoeba's user interface is the industry standard X Window System (X11R5). For X servers running on workstations, a special version of X is available that uses the Amoeba RPC for high-performance communication. When hard-wired X terminals are used, these can be interfaced using the TCP/IP server.

6.10. Connection to UNIX

A special UNIX driver is provided with Amoeba that can be linked into a SunOS 4.1.1 (or higher) UNIX kernel, allowing UNIX programs to communicate with Amoeba programs. It is also possible, as stated before, to use TCP/IP for this communication (e.g., for non Sun machines), but the feature described here is much faster and less complex if Sun workstations are available. Utilities are provided to transfer files between UNIX and the Bullet file server.

7. NONTECHNICAL ASPECTS OF AMOEBA

7.1. Source Code Availability

All Amoeba distributions contain the entire source code. Binaries for the supported machines are also included.

7.2. Amoeba is Unencumbered by AT&T Licensing

Amoeba is a completely new operating system. Although it provides a partial UNIX emulation, it contains no AT&T code whatsoever. Furthermore, the utility programs it comes with have either been written from scratch or obtained from third parties under favorable conditions. Although customers are required to agree to our license, no additional licensing is needed for Amoeba.

7.3. Documentation

Amoeba comes with over 1000 pages of documentation. It is organized in several volumes:

- A collection of published scientific papers describing the basic ideas.
- A users' guide (how to work with Amoeba; man pages for the utility programs).
- A programmers' guide (writing clients/servers; man pages for library routines).
- A system administrators' guide (how to operate and maintain Amoeba).
- Release notes (bibliography, changes, bug information, etc.).

All the documentation is freely available by anonymous FTP from the following sites. See the *README* files there for further details.

Location	Site	Name	Internet address	Directory
Europe	VU	ftp.cs.vu.nl	192.31.231.42	amoeba
USA	UCSC	ftp.cse.ucsc.edu	128.114.134.19	pub/amoeba

7.4. Machines on which Amoeba Runs

Amoeba currently runs on the following architectures:

- Sun 4c and MicroSPARC SPARCstations
- Intel 386/486 (IBM AT bus)
- 68030 VME-bus boards (Force CPU-30)
- Sun 3/60 & Sun 3/50 workstations

7.5. Configuration Required

Amoeba is a heterogeneous distributed system. Although in theory work can be done on a single machine, in practice more than one machine is required. We recommend at least five machines: a file server, a workstation and 3 pool processors. The more pool processors the better. Any combination of of the supported machines can be used.

- **Minimum configuration for a SPARCstation system:**
  File server: ≥ 16 MB RAM, a 300 MB disk, a SCSI tape drive.
  Workstation: ≥ 8MB RAM, monitor, keyboard, mouse.
  Pool processor: ≥ 8 MB RAM.

- **Minimum configuration for 386 or 486 (AT bus) systems:**
  File server: ≥ 16 MB RAM, a 300 MB disk, 3.5" floppy drive, Ethernet card, VGA card, keyboard, monitor, mouse.
  Workstation: ≥ 8 MB RAM, Ethernet card, VGA card, keyboard, monitor, mouse.
  Pool processor: ≥ 4 MB RAM, 3.5" floppy drive, Ethernet card.

  Supported Ethernet cards: SMC/WD 8013, NE 2100, NE2000, 3Com 503

- **Minimum configuration for a Sun 3/60 system:**
  File server: exactly 12 MB RAM, a 300 MB disk, a QIC-24 tape drive.
  Workstation: ≥ 4 MB RAM, monochrome monitor, keyboard, mouse.
  Pool processor: ≥ 4 MB RAM.

Amoeba is normally distributed by FTP, Exabyte tape, QIC-150 or QIC-24 streamer tape. The distribution is about 120 MB of source, documents and binaries. For each architecture a different subtree will be generated, and each binary tree will need another 80 MB. The large size is due to the X libraries compiled into the binaries. The X sources are not included. However, the changes needed for the X11R5 sources are provided.

For embedded applications, where the file server is not necessary and only the kernel is being used, it is possible to run the Amoeba kernel on a single CPU. Some installations are running Amoeba in kernel-only mode, in effect using it as a distributed high-performance kernel for industrial process control applications.

7.6. Pricing

Amoeba is available free to universities that have FTP access to the Internet, and for $500 on Exabyte tape or $800 on QIC-24 or QIC-150 tape to those that do not. To corporate, government, and other users, ACE provides Amoeba at competitive prices.

7.7. Support

Amoeba 5.2 is provided to universities on an ‘‘as is’’ basis, with no support. Although Amoeba is still an experimental system, rather than a production quality polished product (e.g., the UNIX emulation is not 100% complete), it can still be highly useful to anyone interested in distributed systems.

On the other hand, government, corporate, and other users who need full support, including training, enhancements and site-specific modifications can purchase them from ACE. Interested parties are requested to contact ACE for details.

7.8. Ordering Procedure

Amoeba is copyrighted software. It is available free to universities under a ‘‘shrink-wrap’’ license in which the university agrees to use Amoeba only for educational and research work, to not hold us legally responsible for the consequences of bugs in Amoeba, and all the usual stuff lawyers think of. To FTP Amoeba you must agree to the conditions of the license before you can FTP anything other than the license. Copying Amoeba without agreeing to the license is a violation of copyright law. Amoeba can be FTPed from *ftp.am.cs.vu.nl*. Note that your FTP client must support the *quote* command to be able to FTP Amoeba.

University customers not having FTP access and all nonuniversity customers must sign a commercial licensing agreement in which their rights and obligations are explicitly spelled out. To obtain a license, please contact us by email or FAX, being sure to include the postal address to which the license is to be sent. Licenses **cannot** be sent by FAX or electronically.

Customer type	Contact	Email address	FAX number
Universities	VU	amoeba-license@cs.vu.nl	+31 20 6427705
All others	ACE	amoeba@ace.nl	+31 20 6750389

Once the signed license is returned, universities will be sent the tapes. The two tape options include one printed set of the manuals per order. This is exactly the same documentation available by FTP, as described in Sec. 7.3 above. Commercial customers get a comprehensive package, and can order full software support.

8. SUMMARY

Amoeba is a modern distributed operating system that is designed for an environment consisting of multiple computers. Its major properties are summarized as follows:

Amoeba 5.2 Architectural Features

- Transparent distributed computing using a large number of processors
- Parallel computing supported as well as distributed computing
- Microkernel + server architecture
- High performance RPC using the FLIP protocol
- Reliable, totally-ordered group communication
- Support for heterogeneous systems (e.g., Sun-3 and 386 on the same net)
- Can communicate over any mix of LANs, WANs, and shared memories
- Automatic, transparent network configuration

User-level Software

- Object-based
- Multiple threads per address space
- File and directory servers provided (including automatic file replication)
- X windows, release 5 supported
- TCP/IP supported
- ANSI C, Pascal, FORTRAN 77, and Modula 2 compilers and libraries provided

- Language for parallel programming (Orca) provided

UNIX
- Good integration with existing UNIX systems
- Amoeba can talk to UNIX via TCP/IP
- Driver available for Sun UNIX to let UNIX use fast Amoeba RPC protocol
- X terminal can have both UNIX and Amoeba windows simultaneously
- Partial UNIX emulation available (good enough to run the MMDF mail system)
- Over 100 UNIX-like utilities available

Commercial Aspects
- Full source code (except the compilers) is included in all distributions
- No AT&T license is required
- Full commercial support available via ACE (Dutch R&D software company)
- Amoeba has already been licensed by major organizations in Europe and Japan

Weak Points In Amoeba 5.2
- Over 1000 pages of documentation supplied
- Not binary compatible with UNIX
- No virtual memory (for performance reasons)
- Works poorly when there is insufficient memory
- No NFS support
- While fine for experimenting, it is not a totally polished production system

9. ACE - ASSOCIATED COMPUTER EXPERTS BV

ACE has commercialized and supported Amoeba since 1990.

9.1. Technical Background of ACE

ACE is a research and development company that provides leading edge technology, products and services in the areas of: CASE, Operating Systems, Compiler and Networking technology.

ACE technology and expertise include:

- UNIX products: the X/OPEN XPG compliant ACE UNIX and more recently a multiprocessor implementation for computer systems with a heterogeneous multiprocessor architecture
- Compiler products: the integrated ACE EXPERT Compiler family for ANSI-C, K&R C, C++, Parallel C, Pascal, Fortran-77, Fortran-90 and Modula-2 for architectures such as Motorola MC68K, Intel I386, SPARC, T800 and T9000
- Standardization of Operating Systems and Programming Languages through participation in the International Standards Organization (ISO) for C, Pascal, Modula-2, Prolog and ‘‘Language Independent Data Types’’, the American IEEE POSIX commission for UNIX, and the European standardization body CEN/CENELEC for ‘‘European Workshop on Open Systems’’ (EWOS)

- Verification of Operating Systems and Programming Languages through X/OPEN XPG testing, validation of Pascal, Modula-2, Fortran-77 and Cobol with industry-standard validation suites, and validation of ANSI-C through SuperTest (a product developed by ACE in cooperation with HCR in Toronto)
- Computer Aided Software Engineering: due to the heavy requirements for systems software, such as reliability, adherence to industry standards and efficiency, in combination with the complicated nature of this technology, CASE functionality was implemented to integrate Configuration Management, Software Engineering and Quality Assurance in one CASE environment: CADESE (Computer Aided Distribution Environment for Software Engineering).

From a marketing point of view ACE products and technology are mostly distributed via OEM-partners and VARs that integrate ACE technology with their own products, and distributors that resell the product for their particular market.

9.2. Project Background of ACE

In order to strengthen the technical, financial and political backing of its research and development projects, ACE established cooperation with other partners in Europe.

9.2.1. ESPRIT

At this moment ACE participates in several ESPRIT projects. These projects are performed under contract with the Commission of the European Communities (CEC) and involve cooperation with several companies with different backgrounds, objectives (either research or product oriented), culture, language, etc. Setting up a Software Engineering Environment and a Project Control Environment for distributed environments like this is part of ACE expertise today.

10. BIBLIOGRAPHY

1. Kaashoek, M.F., Renesse, R. van, Staveren, H. van, and Tanenbaum, A.S.: "FLIP: an Internetwork Protocol for Supporting Distributed Systems," *ACM Trans. on Computer Systems* vol 11, pp. 73-106, Feb. 1993.

2. Kaashoek, M.F., Tanenbaum, A.S., Verstoep, K.: "Using Group Communication to Implement a Fault-Tolerant Directory Service," *Thirteenth Int'l Conf. on Distributed Computing Systems*, IEEE, pp. 130-139, 1993.

3. Kaashoek, M.F., Tanenbaum, A.S., and Verstoep, K.: "Group Communication in Amoeba and its Applications" , *Distributed Systems Engineering J.* vol. 1, pp. 48-58, July 1993.

4. Tanenbaum, A.S., Kaashoek, M.F., and Bal, H.E.: "Parallel Programming Using Shared Objects and Broadcasting," *IEEE Computer* vol. 25, pp. 10-19, Aug. 1992.

5. Kaashoek, M.F., Tanenbaum, A.S., and Verstoep, K.: "A Comparison of Two Paradigms for Distributed Computing," *Proc. Fifth ACM SIGOPS Workshop*, Le Mont St. Michel, France, Sept. 1992.

6. Levelt, W.G., Kaashoek, M.F. Bal, H.E., and Tanenbaum, A.S., " A Comparison of Two Paradigms for Distributed Shared Memory," *Software Practice & Experience*, vol. 22, pp. 985-1010, Nov. 1992.

7. Tanenbaum, 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. Douglis, F., Ousterhout, J.K., Kaashoek, M.F., and Tanenbaum, A.S.: "A Comparison of Two Distributed Systems: Amoeba and Sprite," *Computing Systems Journal* vol 4., pp. 353-384, Fall 1991.

9. Tanenbaum, A.S., Renesse, R. van, Staveren, H. van., Sharp, G.J., Mullender, S.J., Jansen, J., and Rossum, G. van: "Experiences with the Amoeba Distributed Operating System," *Commun. of the ACM* vol. 33, pp. 46-63, Dec. 1990.

10. Renesse, R. van, Staveren, H. van, and Tanenbaum, A.S.: "Performance of the Amoeba Distributed Operating System," *Software—Practice and Experience*, vol. 19, pp. 223-234, March 1989.

11. Baalbergen, E.H.: "Design and Implementation of Parallel Make," *Computing Systems*, vol. 1, pp. 135-158, Spring 1988.

12. Tanenbaum, A.S., Mullender, S.J., and van Renesse, R.: "Using Sparse Capabilities in a Distributed Operating System" *Proc. Sixth International Conf. on Distr. Computer Systems*, IEEE, pp. 558-563, 1986.