An Introduction to Amoeba

Andrew S. Tanenbaum
Dept. of Mathematics and Computer Science
Vrije Universiteit
De Boelelaan 1081
1081 HV Amsterdam, The Netherlands
Internet: ast@cs.vu.nl

Sape J. Mullender
Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam, The Netherlands
Internet: sape@cwi.nl

1. BACKGROUND

As the 1990s take hold, it is increasingly clear that computer operating systems designed for single processor systems back in the 1970s and 1980s will no longer be appropriate for the needs of the new decade. UNIX† is now almost 20 years old. Although it has gotten much bigger over the years, the basic ideas have not really changed since it was created in the early 1970s. MS-DOS, although not quite as old, is in many ways even less appropriate than UNIX for the powerful computer systems of the 1990s. Perhaps it is time to start over fresh with something new. In this collection of papers we describe the Amoeba distributed operating system, which has been designed and implemented with the technology of the 1990s in mind.

What are the key characteristics of computing now and in the future? We are convinced that two factors will dominate the next decade:

- The need for physically distributed hardware
- The need for logically centralized software

Let us now discuss these in turn.

First, computers are becoming cheaper at an enormous rate. In the 1970s, it was normal for many people to share a single mainframe or minicomputer by running a timesharing system on it. Each user had a terminal with which to access the computer. The ratio of computers to people was very low, often 20 or 50 or even 100 people per machine.

In the 1980s, the personal computer and personal workstation became popular. By the end of the decade, many universities and companies operated using a model in which each person had his or her own machine, all connected by a local area network. The ratio of computers to people became approximately 1 to 1, as many machines as people.

In the 1990s, hardware prices will continue will continue to drop dramatically. We will soon come to a situation in which it is economically feasible to have 20 or 50 or even 100

† UNIX is a trademark of Bell Laboratories.

computers per person. Clearly the current model of giving each person a personal computer or workstation breaks down under these conditions. Nevertheless, the availability of large numbers of powerful single-chip processors is a given. Any system for the 1990s must address itself to the issue of how to deal with a system containing hundreds, if not thousands, of processors, very likely distributed over a considerable area.

The second factor mentioned above is the need for logically centralized software. While it is currently possible to physically connect up a few dozen machines on a local area network, the result is often unpleasant for the users. In many ways, the 1970s model of having one machine that everyone used was in fact much simpler and easier to use than the current personal computer model. With only a little effort, giving each user dozens of computers could produce a complete disaster.

We believe what users want is a system built out of large numbers of powerful microprocessors that take advantage of the current hardware technology, but which together act together in a coherent way that is as easy to use and understand as an old-fashioned timesharing system. Users do not want, and rarely fully understand concepts such as remote mounting, yellow pages, and similar bizarre and complicated things.

We must produce a new generation of operating systems that tie all the pieces together and make the collection of hardware boxes look like a single, integrated machine, rather than a bunch of distinct machines that communicate using some form of network protocol. The user logging into the system should not be aware of how many machines there are, where they are located, what their functions are, where the files are (or how many copies there are), how many processors are needed to run any particular job, or anything else about the physical distribution of the hardware. This is the challenge of the 1990s.

2. THE AMOEBA MODEL

As a first step towards a completely new operating system designed expressly to meet these goals, we began the Amoeba project at the Vrije Universiteit in Amsterdam. Subsequently, the Vrije Universiteit has teamed up with the Centrum voor Wiskunde en Informatica to continue developing Amoeba jointly. In this collection of papers we will describe the Amoeba system and try to show why we think it is appropriate for the coming decade.

Before describing the software, it is worth saying something about the system architecture on which Amoeba runs. The Amoeba architecture consists of four principal components:

- Workstations with a window system for providing user access
- Pool processors for computing
- Specialized servers such as file servers and directory servers
- Gateways to other systems

The key idea here is that the workstations are basically terminals. A typical workstation might be a Sun-3 or an X-terminal. The job of the workstation is to run the window manager and interact with the user via the keyboard and mouse. With few exceptions, such as an editor, heavy computing generally does not occur on the workstations. They are really just glorified terminals.

The computing power lies in the pool processors. In our installation, this consists of a several standard 19-inch equipment racks containing a total of 48 single board computers, each having a powerful CPU (68020 and 68030 at present), 2-4M of memory, and a network connection (Ethernet); diskless workstations could also have been used. When a job needs computing power, it asks the process server to temporarily allocate it some number of processors, which it then uses and then returns. A typical use might be running the *make* program. Suppose *make* discovers that it needs to do 8 compilations, and the compiler has 5 passes. Then 40 processors would be allocated (if available) and all the passes of all the compilations could proceed simultaneously. As soon as they were finished, the processors would go back into the pool to be available for another request by another user.

The specialized servers are machines that need dedicated resources of some kind. The file server runs best on a machine with disks, for example.

Finally, the gateways are used to connect up multiple Amoeba systems at different sites in different cities or even different countries. Their job is to protect the local machines from the wide-area protocols, to make it possible to access a machine in a different city without having to even know that it is distant. The gateways handle all the protocol wrapping and unwrapping transparently.

3. DESIGN OF THE AMOEBA SOFTWARE

The Amoeba software is object-based. The system can be viewed as a collection of objects, on each of which there is a set of operations that can be performed. For a file object, for example, typical operations are reading, writing, appending, and deleting. The list of allowed operations is defined by the person who designs the object and who writes the code to implement it. Both hardware and software objects exist.

Associated with each object is a *capability*, which is a kind of ticket or key that allows the holder of the capability to perform some (not necessarily all) operations on that object. A user process might, for example, have a capability for a file that permitted it to read the file, but not to modify it. Capabilities are protected cryptographically to prevent users from tampering with them.

Each user process owns some collection of capabilities, which together define the set of objects it may access and the type of operations he may perform on each. Thus capabilities provide a unified mechanism for naming, accessing, and protecting objects. From the user's perspective, the function of the operating system is to create an environment in which objects can be created and manipulated in a protected way.

This object-based model visible to the users is implemented using remote procedure call. Associated with each object is a *server* process that manages the object and provides a *service*. Some services are implemented by multiple servers, for added reliability. When a user process wants to perform an operation on an object, it sends a request message to the server that manages the object. The message contains the capability for the object, a specification of the operation to be performed, and any parameters the operation requires. The user, known as the *client*, then blocks. After the server has performed the operation, it sends back a reply message that unblocks the client. The combination of sending a request message, blocking, and accepting a reply message forms the remote procedure call, which can be encapsulated using stub routines, to make the entire remote operation look like a local procedure call.

The Amoeba kernel basically handles communication and some process management, and little else. The kernel takes care of sending and receiving messages, scheduling processes, I/O, and some low-level memory management. Everything else is done by user-level server processes.

There are a variety of standard servers available with Amoeba. These include the file server, the directory server, the process server, the bank server, the boot server, the debug server, the X-windows server, the TCP/IP server, and others.

Amoeba was designed with the idea that a collection of machines on a local network would be able to communicate over a wide-area network with a similar collection of remote machines. The key problem here is that wide-area networks are slow and unreliable, and furthermore use connection-oriented protocols such as X.25, TCP/IP, and OSI; in any event, not RPC. The primary goal of the wide-area networking in Amoeba has been to achieve transparency without sacrificing performance. In particular, it is undesirable that the very fast Amoeba local RPC be slowed down in any way due to the existence of wide-area communication. Based on actual performance measurments, we believe this goal has been achieved.

The basic idea is to have server and client agents running on all the local networks, normally on the gateway machines. When a network wants to import a service from the outside world, it creates a server agent locally that represents the server to the local clients. When a local client does a remote procedure call, the server agent catches the call, creates a client agent on the server's network, and forwards the call to the newly created remote client agent. This client agent then makes an ordinary remote procedure call to the server. Both the client and server are talking to local processes, so they do not notice that anything strange is going on. Only the two agents have to be aware that remote communication is taking place. Processes not using the wide-area network are not affected by it, and can operate at full speed using Amoeba RPC.

A variety of utilities, languages, and applications have been implemented on Amoeba. These include a UNIX emulation package, a parallel version of *make*, and a new language, *Orca*, expressly designed to ease the task of parallel and distributed programming on systems like Amoeba.

4. SUMMARY OF THE PAPERS IN THIS COLLECTION

This book is simply a collection of papers that have been written about Amoeba by the VU and CWI over the course of the past several years. It is divided into 10 sections, as follows:

- Introduction
- Overview of Amoeba

- Design Aspects
- Performance
- Amoeba Over Wide-Area Networks
- Multiprocessor Amoeba
- Broadcasting
- Distributed Programming
- Applications
- Theory

We will now briefly describe each section.

This is the Introduction. The next section contains three overview papers. The first two are specifically about Amoeba, and describe the current (1990) state of the system. The third one provides general background information about the context in which distributed systems like Amoeba are embedded.

The Design Aspects section contains three papers about the Amoeba design, covering the use of capabilities, the file server, and the stub generator.

Amoeba was designed to be extremely fast on a local area network and to provide a very fast file system. In the next section we present some measurements of Amoeba's performance. To the best of our knowledge, both the RPC time and the performance of the file server are better than those of any other distributed operating system yet reported in the literature (for its class of hardware, of course; a system running on hardware that is three times as fast may well do better).

Although Amoeba was originally conceived for use on local area networks, it has also been extended to wide-area networks. The two papers on wide-area Amoeba describe how this was done, based on our experience connecting Amoeba sites in The Netherlands, England, and Norway.

Amoeba has also been run on a true shared-memory multiprocessor. Some prelimary results are given in the next section, primarily emphasizing the memory contention problem.

The basic communication paradigm in Amoeba is RPC. Nevertheless, for some applications, a many-to-one communication is highly desirable. The paper by Kaashoek et al. presents a simple, but extremely fast algorithm, for doing reliable broadcasting. This algorithm has already been implemented. A description of the algorithm and its performance are given here. It is already being used to implement a distributed programming language.

Since Amoeba has been designed to run on a large collection of machines, one of the obvious things to use it for problems in which multiple processors work together on the same problem (parallel processing). We have devised a new model for such computation, and designed a language called *Orca* that implements this model, called the *shared data-object model*. The five papers in the section on Distributed Programming describe the model, the language, its implementation, and some measurements of its performance.

The Applications section describes some other applications of Amoeba, including a fast parallel version of *make*.

Finally, as well as designing and implementing a new distributed operating system, we have also done some theoretical work in this area. The two papers in this section deal with the theory of locating servers in a decentralized way, and handling file replication in a decentralized way.

When citing any of these papers in the literature, please refer to the original papers, rather than this (nameless) collection. The complete bibliographic information for each paper is given in the table of contents. It is permitted to copy this collection of papers in whole or in part for educational or research purposes.

5. AVAILABILITY OF AMOEBA

It is our intention to make Amoeba widely available to the educational and research community in source form, much as UNIX was in the early days. Amoeba currently runs on Sun-3s and VAXstations. A port to the Intel 80386 is well underway and will be completed shortly. Ports to RISC machines are being considered. In a typical configuration, some machines would be allocated as user workstations, others as file servers, and still others as pool processors. The file server and pool processor machines need not have keyboards or displays, allowing less expensive "server" configurations to be used.

Amoeba has been written entirely from scratch and is provided with source code (written in C). Neither the kernel nor the utilities contain any AT&T or Berkeley code whatsoever, so AT&T and Berkeley licenses are not needed.