An Overview of the Andrew File System

John H Howard

Information Technology Center
Carnegie Mellon University

ABSTRACT

Andrew is a distributed computing environment being developed in a joint project by Carnegie Mellon University and IBM. One of the major components of Andrew is a distributed file system. The goal of the Andrew File System is to support growth up to at least 7000 workstations (one for each student, faculty member, and staff at Carnegie Mellon) while providing users, application programs, and system administrators with the amenities of a shared file system. This overview describes the environment, design, and features of the file system, and ends with a summary of how it "feels" to end users and to system administrators.

1. History and Goals

The Information Technology Center was started in 1982. It grew out of a task force report on the future of computing at Carnegie Mellon, which recommended that the University move to a distributed computing system based on powerful personal workstations and a networked "integrated computing environment" to tie them together. Overall project goals included advancement of the state of the art in distributed computing, establishing a new model of computer usage, and providing sufficient accessibility and functional richness to become an integral part of users' lives. The integrated computing environment was to allow local control of resources but to permit easy and widespread access to all existing and future computational and informational facilities on campus. Network and file system objectives thought to be necessary to achieve these general goals included high availability, ease of expansion and of incorporation of new technologies, and good performance even during peak periods, with graceful response to overload.

As the project developed, five focal areas emerged: the workstation hardware and operating system, the network, the file system, the user interface, and the message system. This overview deals with the file system, with some passing references to the workstation hardware and the network.

It was generally agreed from the beginning that a high-function workstation should be used rather than a more economical but less functional tool such as a personal computer. The most important feature was a million pixel bit-mapped display; other characteristics (such as a 1 million instruction per second processor and a million bytes of main memory) were selected in proportion to the display's needs. The decision to use UNIX,† (specifically 4.2BSD) as an operating system was made early in the project, based its availability, the desire to use standard tools wherever possible, and the presence in 4.2BSD of desirable features such as virtual memory and networking support.

The network consists of a mixture of Ethernets and Token Rings, tied together by bridges (sometimes referred to as routers). It was built up gradually by inter-connecting pre-existing departmental Ethernets and gradually reorganizing the overall topology into a hierarchy, which at present consists of a campus backbone and a large number of local nets in the individual academic buildings. The local nets are connected to the backbone by fiber optic links which run from the building entry wiring closets to the University Computing Center building. The routers and the backbone are physically located entirely in

† UNIX is a trademark of Bell Laboratories.

the UCC building (and one other important academic building), thus greatly simplifying fault isolation and repair.

Given the choice of 4.2BSD, it was easy to select the DARPA TCP/IP protocols as the campus standard. While TCP/IP was not the majority choice of any constituency at Carnegie Mellon, it was at least available for most of the existing computers and operating systems. Other protocols are tolerated on many of the network segments, but the routers operate at the IP packet level.

2. File System Design

The general goal of widespread accessibility of computational and informational facilities, coupled with the choice of UNIX, led to the decision to provide an integrated, campus-wide file system with functional characteristics as close to that of UNIX as possible. We wanted a student to be able to sit down at any workstation and start using his or her files with as little bother as possible. Furthermore we did not want to modify existing application programs, which assume a UNIX file system, in any way. Thus, our first design choice was to make the file system *compatible with UNIX at the system call level.* While we realized that there would be some unavoidable differences in performance, in failure modes, and even in some functions, we were determined to minimize them wherever possible. The few differences will be discussed later.

The second design decision was to use *whole files* as the basic unit of data movement and storage, rather than some smaller unit such as physical or logical records. This is undoubtedly the most controversial and interesting aspect of the Andrew File System. It means that before a workstation can use a file, it must copy the entire file to its local disk, and it must write modified files back to the file system in their entirety. This in turn requires using a local disk to hold recently-used files. On the other hand, it provides significant benefits in performance and to some degree in availability. Once a workstation has a copy of a file it can use it independently of the central file system. This dramatically reduces network traffic and file server loads as compared to record-based distributed file systems.[SOSP11] Furthermore, it is possible to cache and reuse files on the local disk, resulting in further reductions in server loads and in additional workstation autonomy.

Two functional issues with the whole file strategy are often raised. The first concerns file sizes: only files small enough to fit in the local disks can be handled. Where this matters in our environment, we have found ways to break up large files into smaller parts which fit. The second has to do with updates. Modified files are returned to the central system only when they are closed, thus rendering record-level updates impossible. This is a fundamental property of the design. However, it is not a serious problem in the university computing environment. The main application for record-level updates is databases. Serious multi-user databases have many other requirements (such as record- or field-granularity authorization, physical disk write ordering controls, and update serialization) which are not satisfied by UNIX file system semantics, even in a non-distributed environment. In our view, the right way to implement databases is not by building them directly on the file system, but rather by providing database servers to which workstations send queries. In practice, doing updates at a file granularity has not been a serious problem.

The third and last key design decision in the Andrew File System was to implement it with *many relatively small servers* rather than a single large machine. This decision was based on the desire to support growth gracefully, to enhance availability (since if any single server fails, the others should continue), and to simplify the development process by using the same hardware and operating system as the workstations. On the other hand, we are interested in experimenting with a larger server, possibly based on a mainframe, to see if it were more cost-effective or easier to operate. At the present time, an Andrew file server consists of a workstation with three to six 400-megabyte disks attached. We set (and have achieved) a price/performance goal of supporting at least 50 active workstations per file server, so that the centralized costs of the file system would be reasonable. In a large configuration like the one at Carnegie Mellon, we also use a separate "system control machine" to broadcast global information (such as where specific users' files are to be found) to the file servers. In a small configuration the system control machine is combined with a (the) server machine.

3. Implementation

The Andrew File System is implemented in several parts, which are most conveniently explained by following what happens when a application program attempts to open a file.

The file open system call, issued by the application program, is intercepted by a small "hook" installed in the workstation's kernel. The user's program is suspended and the intercepted request is diverted to a special program, implemented as a user-level process on the workstation. This program, the Andrew Cache Manager, forwards the file request to a file server, receives the file (if everything went well), stores it on the local disk, and sends back an open file descriptor via the kernel to the user's program. The user's program then reads and writes the file locally, without any special help from the kernel, the cache manager, or the file server. Eventually, the close operation is also diverted to the cache manager, which (if the file was modified) sends the updated copy back to the file server.

The Andrew Cache Manager has several other important functions. It keeps copies of recently used files on the workstations' local disk, thus minimizing network traffic for frequently-reused files. It also caches directories and status information and, in fact, performs the entire directory lookup function, again reducing server loads. It maintains much of this status information across file server or network failures and even workstation restarts, simply re-validating cached information the first time it is used. (Once a file has been validated in this way, the server will notify the cache manager if the file changes.) Most of the burden of reproducing UNIX system call semantics falls upon the Andrew Cache Manager as well.

The file server consists of both hardware and software. The software consists primarily of a "file manager" program which runs in a single UNIX process and is internally multiprogrammed by a "light-weight process" package which is capable of handling several requests in parallel.

The operations performed by the Andrew File Manager are reading, creating, or replacing files, reading directories, setting and releasing advisory locks, and various administrative operations. The manager does the following things to handle a typical operation:

verify the identity of the user.

check the user's permission to perform the operation.

reserve disk space, in the case of an update operation.

perform a bulk transfer of the requested file to or from the workstation as necessary.

replace the old version of an updated file with the new version atomically.

send an ending status to the workstation.

keep a record of the workstation's interest in the file, for use in future update notifications.

Supporting programs include an "authentication manager", an "update manager" and a "status manager". The authentication manager maintains a database of users and authenticates them. The update manager keeps software and internal databases up to date and consistent in configurations with more than one file server. The status manager serves as a centralized collection point for information on the status of the various file servers. This strategy makes it possible for anybody to make occasional or even periodic status queries without overloading the actual file servers.

4. Features

Here are several specific aspects of the file system worth some more description.

4.1. Authentication

When a user logs into a workstation, the workstation goes through an authentication procedure based on the user's password. By authenticating in the file server we avoid depending on the workstation kernel, which is exposed to user modification, for authentication. Although undue reliance on the security of network neighbors is not the only security weakness of distributed systems, we feel it is an important one.

The authentication mechanism is automatic as far as users are concerned. It involves an exchange of encrypted messages between the login program and the authentication manager. A successful exchange establishes the identity of both the user on the workstation and of the authentication manager. It produces some "tokens" which the Andrew Cache Manager uses to establish connections to file servers as it needs them. These tokens can be passed on to other workstations in order to support remote login capabilities. Tokens time out after 25 hours in order to provide some control over the possibility of their being stolen.

Two new utility commands have been provided to make authentication more convenient: *log* reauthenticates a user (for example if the tokens have timed out), and *unlog* discards the tokens, thus rendering the workstation safe for use by other individuals. It is common to execute an *unlog* command when logging out from a workstation.

4.2. Access Lists

We felt that the traditional 12-bit access mode flags of UNIX were inadequate for a campus-wide file system, so we supplemented them by an access list mechanism. An access list specifies various users (or groups of users) and, for each of them, specifies the class of operations they may perform. Access lists are associated with directories. The operations they specify are:

read any file in the directory

write (update) any file in the directory

insert new files in the directory

delete files from the directory

lookup files in the directory

lock files in the directory

administer the directory (that is, change the access list)

The standard UNIX bits are retained as well, with both conventional and access list permission required to operate on a file. Thus the standard commands such as *chmod* work as specified (although with somewhat dubious security.) The super-user has no special privileges as far as the access list mechanism is concerned. This leads to some difficult issues in the handling of the "setuid" mechanism. Although we have been able to work around the problems we have encountered, this is a significant divergence from standard UNIX semantics.

Access lists are displayed and changed by a user command, *fs*, which passes its requests through the cache manager to the file managers. The *fs* command also provides miscellaneous other operations including listing and controlling the user's disk space quota and other properties of the logical volume (see below), flushing files from the cache, finding out file locations, and general administrative functions.

4.3. Logical Volumes

The Andrew File System groups files into aggregates called "logical volumes". A typical logical volume would be a single user's files, or a particular release of the system binary files. Logical volumes are interconnected by a "mount point" mechanism resembling standard UNIX mounts. End users seldom need to be aware of logical volumes. On the other hand, operators and system administrators are almost exclusively concerned with logical volumes rather than with individual user files, as the logical volume is the unit for operations like backup, load and space balancing between file servers, and redundant storage of read-only files.

An important file system operation from an operator's point of view is the creation of a read-only snapshot, or "clone", of any logical volume. This is implemented in such a way that it is quick and inexpensive to make clones, so they are used in several important ways. First, new software releases are typically made by cloning the system binaries. Read-only clones can be replicated on multiple servers, leading to significant increases in availability and performance. (At Carnegie Mellon, there are three servers devoted exclusively to read-only clones of system volumes.) Second, individual users'

logical volumes are cloned every midnight. The main reason for this is to get a consistent snapshot which can then be backed up over the next day. A secondary benefit is that the clone is made available to each user under the name "OldFiles", so that all the user's previous day's work is available in case of a blunder. This significantly reduces requests to recover files from the backup tapes.

5. End Users' Viewpoint

The key fact about the Andrew File System from an end user's viewpoint is that it closely resembles a standard UNIX file system, yet allows the user to sit down at any campus workstation and get at a uniform set of files. In almost all cases it is possible to ignore the distributed implementation, the automatic transfer of files on demand, and the caching mechanism. Application programs imported from more centralized systems can be run without any modification whatsoever and will almost always work. The rest of this section will talk about some differences, but the reader should remember that the differences are exceptions rather than the rule.

The main differences end users see relate to performance and failure modes. There is a noticeable wait when the workstation must fetch a large file. (At the present time the effective transfer rate is about 50 thousand bytes per second, so a million-byte file takes 20 seconds.) Although in absolute terms this is reasonably fast, it is still noticeable and sometimes annoying.

Failure modes in a distributed system are intrinsically different from those in centralized systems. The file system at Carnegie Mellon suffers from occasional failures in the file server hardware, the network connecting workstations to the file servers, and (occasionally) in the power or cooling supply to the entire computing center facility, or the entire campus internet. The design decision to implement the file system with multiple servers is rendered useless when all the servers are disabled by a power or cooling failure, as has happened several times. Much more common, however, are individual server failures. When the cache manager loses contact with a file server it goes through a one-minute timeout, after which it switches to a disconnected mode in which it allows cached files from that server to be read, but not written. If the failed server has your personal files, it won't be long before you get stuck on missing file or a write. In a few cases it is desirable to modify critical application programs, like text editors, to deal gracefully with the resulting error conditions. For example, it is more common for the *close* operation to fail than it is in a conventional UNIX system.

Functionally, the main difference in the Andrew file system is its use of access lists, which we feel have significant advantage in both flexibility and understandability from a user's standpoint.

6. Operators' Viewpoint

Unlike the user, the operator sees the Andrew file system as being very different from a standard UNIX file system. In fact, the main point of contact between them is that UNIX is used as the vehicle for running the file servers. Operators do not (and in fact can not) view or manipulate individual files directly when logged into file servers. (They can, of course, get at individual files through workstations if the access lists allow them to.) Instead, operators and system managers see the file system as a large collection of "logical volumes". Typical operations they perform are to create, adjust, clone, move, and back up logical volumes. Most of these operations are highly automated. For example, there is an operator command *move-vol* which moves a volume from one file server to another. This can be done even when the volume is in use without significant disruption (there may be a period of a few seconds during which the cache manager reports to the user that the volume is busy.)

An interesting question is that of "operating" individual workstations. The Andrew file system is neutral in this respect. There exist at least two strategies, one very centralized and one very distributed. The centralized strategy, which is used for most Andrew workstations today, is heavily dependent on the file system. Individual workstations have a minimum of locally-stored files, such as the UNIX kernel itself or the password list. When a workstation is rebooted a special utility program selectively updates these few files from the central file system. The ordinary user doesn't know the super-user password, and all workstation software is kept up-to-date by system administrators by updating the central directory. At the opposite extreme is the strategy in which the workstation has a full set of local files, maintained entirely by the user, but also uses a connection to the file system for some shared

directories. This decreases dependency on the file central system, but fails to take advantage of their benefits such as automatic backup and access to new software releases; it also requires a significantly larger local disk.

A recent development in the Andrew file system is the introduction of a "cellular" mode whereby subsets of the file servers can be administered separately and still be interconnected. The different administrative units may even have separate user registration lists. This feature provides a new dimension of departmental autonomy to the distributed system.

7. References

[SOSP11] *Scale and Performance in a Distributed File System*, by J.H.Howard, M.L.Kazar, S.G.Menees, D.A.Nichols, M.Satyanarayanan, R.N.Sidebotham, and M.J.West, in *Eleventh Symposium on Operating Systems Principles*, Austin, Texas, November 1987. Will appear in a future issue of the **ACM Transactions on Computer Systems.**