Design of the Sun Network File System

Bill Joy

Company Confidential

ABSTRACT

This paper summarizes the issues in the design of the Sun Network File System. It compares the approach chosen to other UNIX-based file systems which exist or are proposed. Major components of and the state of the implementation are described.

Current UNIX Facilities

Sun uses the version of UNIX with the most advanced networking facilities, 4.2BSD. This version provides transport protocol support for the DARPA standard internet protocols, TCP and IP. The user interface was designed to be transport protocol independent, so that other protocols may be easily accomodated. The user-level network-oriented facilities provide access to both datagram and virtual circuit services, allowing users to build network services as UNIX applications processes.

Sun has enhanced the basic set of protocols supplied with 4.2BSD with a remote disk access protocol, called ***nd***, or network disk. This protocol allows a number of diskless workstations to operate, where the workstations share a central disk drive by partitioning it into a number of areas. The network disk protocol operates by providing the UNIX system with a new "device-driver" which implements a virtual disk drive. The UNIX file system code runs in each workstation, and each workstation uses ***nd*** to simulate a disk drive which is private to the workstation. Only read-only files may be shared between workstations.

Although ***nd*** lacks facilities for sharing data, it does have some properties which are desirable: it allows distributed caching of information remote from the central disk storage, and workstations can continue to operate even after the server crashes and reboots. The protocol used is relatively efficient; performance limitations of the protocol setm largely from the relatively low performance of the Ethernet interfaces available on the Sun-1 Models 100 and 150.

Goals for the Network File System

The most pressing need for a Network File System arises from the lack of shared file access in the current ***nd*** protocol. Without the ability to share files, users must exchange files using a file transfer command (rcp) or by using a server which executes remote commands (rsh). This is not convenient.

We feel that the most important requirements for the Network File System are high-performance, reliability, transparency, and ease of network administration.

To be high-performance the file system should have both low latency and high throughput. We have built a special Ethernet interface which has a large amount of on-board buffering and a page-map to rearrange 1024-byte pages into larger blocks for transfer to and from a disk. This

Ethernet will be used in Sun Network File Server nodes to increase the throughput onto the Ethernet, by offloading the host cpu from packet processing, using the Intel Ethernet chip. The special Ethernet server architecture, when combined with the high-performance 4.2BSD file system should yield very high input/output rates to and from the server.

To achieve low-latency it is very desirable to allow distributed read-ahead and caching. Recent measurements have shown that in a typical UNIX system over 90% of the read-ahead blocks are used. By overlapping computation with pre-fetching of the next block to be sequentially read from a file system idle time is greatly reduced, and the total system throughput is greatly increased. With distributed caching comes the problem of stale data in the caches.

Reliability of the file system derives primarily from the robustness of the 4.2BSD file system. In addition, the File Server Protocol is designed so that client workstations can continue to operate even when the server crashes and reboots. This property is shared with the current ***nd*** protocol, and has proven to be quite desirable. We achieve continuation after reboot without making assumptions about the fail-stop nature of the underlying server hardware..

Transparency of the network file system is very important. We do not wish to modify existing UNIX applications, or to make major restrictions in the way users view file systems which are stored on the network. To this end, the Sun Network File Server provides file systems which are interchangeable with file systems in local disk partitions. Each workstation constructs its file system hierarchy from all the file systems available on the network, just as the current UNIX release constructs its file system tree from the file system partitions on the local machine. No restrictions are placed on the way in which these file systems are named or mounted. While conventions will normally be used to make the set of file systems used by a group of machines the same, the use of experimental configurations or protection considerations may cause the set of file systems used by different workstations to differ.

Finally, ease of network administration is felt to be a paramount concern. Large networks can become quite complicated, and administering these large netorks can be very time consuming. We wish to make sure that a set of network file systems is no more difficult to administer than a set of local file systems on a timesharing system, for which UNIX has evolved a convenient set of maintenance commands over a period of years. To this end, the file systems on the network are mounted, dumped and otherwise maintained in a manner similar to local file systems. For a workstation to gain access to a newly created network file system all that is necessary is to add a single command to the initialization sequence executed by the workstation at boot time. Since this initialization script can easily be shared by all the workstations on a local network, adding a file system and making it available to all workstations is an easy prospect.

Layers in the current UNIX file system

The current UNIX file system implementation is logically divided into four layers. The topmost layer maintains a ***file*** table, containing seek pointers into the files being accessed. Each entry in the file table points to an ***inode*** structure. The ***inode*** is the central data structure in UNIX, and each file, directory or device is represented by an ***inode***.

Operations on ***inodes*** include reading and writing, and truncating the underlying file, and changing file attributes such as ownership or access mode. Each process has, in addition to its open ***file*** table two additional ***inodes*** which refer to the current working directory of the process and the root directory of the file system. Operations involving path names work relative to these two inodes. Such operations include creation of new files and directories, their removal, renaming, etc.

At the next lower level in the system UNIX implements a pool of input/output buffers. These buffers are used for all reading and writing of data in files as well as operations involving the internal structure of the file system itself. These ***buf*** structures also serve as the input/output control blocks of the system, and as the file systems buffer cache. Read-ahead and delayed writes of file system data (write-behind) occur within this cache.

The fourth, and lowest, level of the file system implementation is the block device level. This is the level at which the ***buf*** implementation interfaces to disk drivers, and also to the network disk driver. Each block device supports reading and writing on a linear array of blocks. The underlying device driver handles any media defects by forwarding or otherwise rerouting requests to spare good locations so that the file system need not worry about media problems.

Design alternatives

In order to maintain consistency of a shared file system, a group of workstations must perform operations in a consistent way. This requirement is critical when considering which of the four layers a shared file system should operate at. A second, also very important, consideration is the degree to which the different systems will be dependent on the fine structure of the file system implementation.

We believe that UNIX will exist in a heterogeneous environment where there will be other operating systems and/or other file servers on the network. If the other operating system are not UNIX systems, then some protocol translation may be necessary for UNIX to have access to the files there, but we believe that transparent access to the files is very desirable.

While it would be possible to add a distributed locking mechanism to the current ***nd*** protocol so that it could be used to access shared files, this is undesirable in that it begs the question of how other file systems (such as a MS/DOS file system or a VAX/VMS file system) could be integrated into the network system transparently. The use of a very low-level protocol also removes from the province of the file server control over the integrity of the file system contents.

The new version of VAX/VMS has a distributed file system implemented in this way, where each node on the network has a copy of the file system code (like the current Sun 1.0 release), and each node can write directly into the shared disk area, on the underlying disk representation. This stems from the use, in the VAX architecture, of minimally intelligent, but nevertheless autonomous, disk controllers. While this organization is suitable for a centrally administered set of machines which share a common high-speed bus (e.g. the CI), it is an undesirable orgainization for a large set of workstations on a local network, where responsibility for the file server integrity is much more naturally left to the server itself. This suggests that the new Sun File Server Protocol should not operate at the same level as the ***nd*** protocol.

In order to allow distributed caching, we wish to use a level above the logical ***buf*** level to implement the file service. We expect that the function of caching information, currently shared between the ***buf*** routines and the virtual memory routines, will be centralized into a generalized virtual memory mechanism incorporating copy-on-write virtual memory in a future release of the system. In either implementation the supplied buffer cache is available for the use of the client network protocol implementors.

Thus we have decided to place the Network File Server Protocol implementation at a high-level in the UNIX hierarchy, by generalizing the ***inode*** level. The new version of ***inodes*** are called ***vnodes***. The ***vnode*** implementation continues the effort, started about two years ago by the author when at U.C. Berkeley, to convert the UNIX system to use object-oriented programming techniques internally. By creating a new abstract data type which has the properties of the current ***inode*** structure, but which hides the details of the concrete implementation from the

client of this abstraction, we can make UNIX independent of the particular implementation of the file system. This allows us to write different versions of the file system, called virtual file systems.

The first two virtual file systems which need to be implemented are the one which replaces the current (concrete) local file system, and one which speaks to the Sun Network File Service. An additional important virtual file system turns operations on a file system into messages on a ***socket*** which can be read by a user process. This is similar to the ***portal*** facility of the 4.2BSD proposal, which was never impelemented.

The local file system implementation in the new version of the system differs mostly cosmetically from the current implementation, but there have been substantial simplifications made since, as the code was rearranged to hide information more cleanly, messy details of the current code became apparent and could be eliminated. One of the benefits of the division between the base-level ***vnode*** code and the implementation of the code for file systems in a local disk partition (on top of the ***buf*** structure) was the separation of the code for handling devices from the code for handling the file system itself. Historically these notions were very heavily intertwined around the ***inode*** notion, and not fully developed as separate concepts. Since it is now possible for a workstation to have some devices but no local code implemeting a UNIX file structure, the divorce of these entangled concepts was critical. The resulting additional clarity is a direct benefit of the use of the object-oriented programming style, and results in a smaller and more maintainable system, with a shorter, and more separate, list of concepts.

While the Sun Network File Service will be the first network file protocol to be adapted into Sun UNIX, and of paramount importance for diskless workstations, it is likely that a great amount of additional energy will be expended in creating additional virtual file system drivers to talk to other file servers and other operating systems. Recognizing this, we have taken great care in the design of the interface between the baseline UNIX system code and the virtual file driver interface. This interface is reminiscent of the current interface between the transport-protocol independent ***socket*** support routines and the underlying suites of transport protocols. We are preparing a document carefully describing the details of writing a virtual file system driver, much as a current documents describe the writing of device drivers and network protocols. This document is planned to be finished in the timeframe of the first bootstrap of the Sun Network File Service, so that other implementors can begin related implementations with due haste.

While other efforts to produce network file systems have attempted to provide efficient or general file server protocols, we believe that the current effort is unique, at least among UNIX network file system implementations, in attempting to create a framework for writing file servers rather than simply creating a single file server. We believe that this will allow the system to be adapted into a number of different networking environments, and to easily conform to whatever file server protocols become important in the UNIX environment, be they defined by AT+T, IBM, Microsoft, DEC, etc.

Facilities omitted

For the first implementation of the network file system it is our goal to provide exactly the current set of UNIX file services to a set of workstations, no more and no less. This means a full hierarchical file system with user and group based protection, and mountable network file systems. We considered, but have for the time rejected, the notion of adding some transaction-oriented facilities to the UNIX facilities. It is possible, using a technique described by Paxton, to implement single-file committs within the current UNIX system.

Multi-file committs and distributed nested transactions are the only more general facility which could be considered. While one company (Locus, Inc.) is attempting to construct a distributed

version of UNIX which supports these facilities, it is our view that they are still a subject of active research and largely the province of specialized data-base systems. We believe that such data base systems will find adequate support within the Sun UNIX system for constructing the mechanisms they need to support nested transactions and multi-file committ on their databases, and would likely not use these facilities even were they provided within the Sun File Service. The additional risk and performance penalties resulting from the added complexity of these facilities, have caused us to decide against including any such facilities, at least in the initial few releases of the Sun File Service.

Another subject of study was the issue of providing replicated files to increase the availability of data on the network. We believe that the proper way in which file replication should be done is still a matter of intense study, and that the state of the art is not up to automatic selection of primary and secondary sites for data storage. The complexity of system-maintained automatic file replication is beyond what is normally chosen for UNIX. We have, instead, opted for a system where there is a single primary site for data, and the client workstations see a service interruption for files when a server becomes unavailable. When the server returns to service, the client workstations can proceed.

The general framework of the Sun system, with its general virtual file system interface, will allow experimentation with both replication and transaction processing. The virtual file system concept supports the construction of a driver which transparently replicates the files on which it operates. A set of *ioctl* calls related to transaction processing could be defined and implemented by a particular virtual file system driver to experiment with such facilities. We believe this is the proper way to experiment with such facilities until the state of the art advances so that a single solution to these problems is recommended and has a simple, efficient and widely-accepted implementation.

Other UNIX File Systems

When the 4.2BSD facilities were being constructed, the author recommended that the group at Berkeley not attempt to construct a file server within the system. At the time, there appeared to be a large number of other groups working on file service for UNIX, and we expected that one of these efforts would produce a workable file service which we could incorporate into the 4.2BSD system. In fact, this has not happened. We spent our effort on other parts of the system, but were disappointed that no file system emerged.

In examining the other file systems again while preparing the current proposal, the author found that these file systems were still either unavailable or unsuitable, to wit:

COCANET — This work, at U.C. Berkeley to provide a transparent distributed file system under 4.1BSD and version 7 UNIX was never completed, as the principal implementor changed his thesis work to a different area, and the sponsoring faculty member joined a small database startup company. Some of the implementation ideas of COCANET are present in the current 4.2BSD *socket* facility and others have been borrowed into the new file server design presented here, but COCANET itself is defunct.

LOCUS — This work at UCLA to provide a version of UNIX supporting transactions and replcated files has moved into industry at a company called Locus, Inc. Sun considered licensing this software about a year ago but the work was not yet finished, and it was clear that at least two years of work would be required to commercialize the software. This is the system which supports nested transactions and replicated files in the operating system. The author consulted experts on database systems known to him to ask whether they would find the facilities

of LOCUS useful in constructing a distributed relational data base. It was their opinion that building such a database on top of the network file service and communications facilities present in the proposed new Sun system was more desirable than using the Locus facilities. If the commericialized Locus is found to be popular in the marketplace, we believe that it can be considered for inclusion in the Sun UNIX system as a particular virtual file system driver.

NEWCASTLE The "Newcastle Connection" is a set of software developed at the University of Newcastle in the United Kingdom. While widely publicized, it is, in fact, little more than a set of library routines which you load with your user program to cause it to use network calls rather than perform local file system accesses. This approach is severely limited in performance because it precludes any form of distributed caching. It also particularizes the file access protocol to a single, user-visible one, and cannot make the existence of different kinds of file systems on the network transparent to the users of the system. Since we firmly believe that such a heterogeneous set of network file systems is likely to exist or to come to exist in most environments where Sun Workstations are used, we believe that this approach is not adequate. In addition, the performance of this approach has never been demonstrated to be adequate, since the hardware used in the prototype system was extremely poor (byte-at-a-time interrupts!).

LUCASFILM Some system programmers at Lucasfilm, Ltd. wrote a network file access protocol which ran as part of 4.1BSD. This protocol was limited in that it did not offer remote current directories, and did not support diskless operation. (To be fair it was designed for use between large VAXes, not between workstations.) Because of the amount of manpower available, the implementation was not cleanly integrated into the UNIX kernel, and thus was not general enough to be considered for use as the general proposal advanced here.

Thus, rather than modifying UNIX to implement one of these protocols, we have designed and coded changes to UNIX to support a variety of file system implementations. By creating this "file-driver" interface, similar in spirit to the UNIX "device-driver" interface, we have insulated the base UNIX system from the detailed implementation of its file system. The only other systems known to the author which have taken as general an approach are message-based. UNIX is a process and monitor-based kernel, and a totally message-based approach to file system support is too inefficient for consideration given our performance goals.

We expect the flexibility of this approach to be called into use fairly soon. It is our expectation that one or more network file service protocols will soon come into use on UNIX-based machines supported by Microsoft and/or IBM. When details of these protocols are made available it will be possible to write a "file-driver" to access these remote file systems from the Sun UNIX, concentrating efforts on the protocol itself, rather than on changing the base part of UNIX. Having to change the base part of UNIX only once should provide a substantial time-to-market advantage for Sun.

Components of the Implementation

The implementation of the Sun Network File Service consists of three major parts. The first part, and the one most nearly completed at this writing, is the modification of the base system to have a virtual file system interface. The document "Sun UNIX Modifications to use the Sun Network File Service" describes the structure of these modifications, comparing the new internals of UNIX to the existing internals. It includes a listing of the current state of the virtual file system ***vfs*** and virtual inode ***vnode*** routines.

In parallel with the creation of the *vfs* and *vnode* interfaces we have created a new version of the local file system code. The local file system code is logically derived from the portion of the file system semantics in the old system which is not now implemented in the new base-level *vfs* and *vnode* code. A major effort here, as noted before, has been to separate the device support and *inode* code in the system, so that diskless workstations can operate with only the device support resident, and the *inode* code can then exist only on machines which support file systems in disk partitions.

A draft version of the Sun Network File Protocol has been written, and is described in the document "Sun Network File Protocol Design Considerations." The local file system implementation as a virtual file system has been carefully modified as part of the splitup of the old file system code so that it may be directly called from the server network virtual file system routine. This leaves the server code to deal with reliability, authentication and protection issues. The server handles these issues and then just calls the same code that an operation would call on the local machine. This is possible because the virtual file system operations are all atomic, and do not require transactions or nested transactions to impelement. This makes a very simple file service implementation possible. Complexity in the server can be reserved for the sake of reliability and performance.

A number of alternatives have been identified for the way in which caching can be done between the server and the client. We plan to choose betwen these alternatives based on actual measurements, rather than choosing in advance of implementation.

Summary

We have reviewed the current state of the UNIX system and discussed considerations in design of a network file service facility. Rather than designing a single protocol into the system, we have seen the advantage of designing a virtual file system interface, having discussed the disadvantages of using lower-level interfaces, such as a device level interface.

Alternative UNIX file systems were examined and all extant file systems were found to be lacking in one way or another. The fact that future important network file systems can be adapted to quickly was seen to be a major advantage of the suggested approach.

We finally described the three pieces of the implementation, the virtual file system and vnode code, and the two halves of the file service implementation. We noted that the file service simply shares the local file system code after dealing with message transport and other network issues. Issues of buffering in the network file system were seen to be very important, and are discussed in a sibling document.