Clue Tables: A Distributed, Dynamic-Binding Naming Mechanism*

Cheng-Zen Yang, Chih-Chung Chen, and Yen-Jen Oyang

Department of Computer Science
and Information Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.

Abstract

This paper presents a distributed, dynamic naming mechanism called clue tables for building highly scalable, highly available distributed file systems. The clue tables naming mechanism is distinctive in three aspects. First, it is designed to cope well with the hierarchical structure of the modern large-scale computer networks. Second, it implicitly carries out load balancing among servers to improve system scalability. Third, it supports file replication and dynamically designates a primary copy to resolve possible data inconsistency. This paper also reports a performance evaluation of the clue tables mechanism when compared with NFS, a popular distributed file system.

1 Introduction

Distributed file systems are the backbone of the modern network computing environment. The naming mechanism in a distributed file system maps the logical name of each individual file to its physical location. In the design of a modern distributed file system, availability and scalability are two essential concerns [1, 2]. In order to build a highly available, highly scalable distributed file system, a designer must incorporate a naming mechanism that can cope with these concerns.

To meet the demands, a naming mechanism must be distributed in nature, and support file replication and dynamic binding. The naming mechanism must be distributed in nature because centralized naming mechanisms suffer low scalability due to limited capacity of the central naming server. The naming mechanism must support file replication because file replication improves both availability and scalability of the system. The presence of replicated file copies prevents service disruption due to failure of a single file server and thus improves system availability. With file replication, the scalability of the system is improved because clients can access replicated file copies on different servers to avoid congestion of a particular file server. To maximize the benefits of file replication, the naming mechanism must support dynamic binding. With dynamic binding, the clients that initially turn to a crashed server for file service can establish new connections on-the-fly to other servers that have replicated copies of the files. Also, clients can dynamically select a server for binding to achieve a good load balancing among servers.

In this paper, we propose a new distributed, dynamic-binding naming mechanism called clue tables. The clue tables mechanism offers the basis to build a highly available, highly scalable distributed file system and is distinctive in three aspects:

1. It is designed to cope with the hierarchical structure of modern computer networks –
   In a modern computer network, particularly a large-scale computer network, bridges are commonly installed to partition the network into a number of clusters. For example, the network in a research institute may be partitioned so that the computers in each laboratory form a local cluster. The hierarchy of network partitions may extend over several levels. The major distinction of the clue tables mechanism is that it was designed to cope with the hierarchical network structure. With clue tables, we can make clients turn first to local file servers to locate a file. If a client can not find the file on local servers, or the local servers that store the file are unavailable, e.g. crashed, then the client will automatically go one level up in the network hierarchy to locate the file on remote servers. The main reason behind

*This research was sponsored by National Science Council of R.O.C. under grant NSC 83-0408-E-002-002

adopting this practice is to make most bindings among clients and servers occur in conformity with network hierarchy design.

In reality, this is the main distinction between the clue tables mechanism and the similar prefix tables mechanism [3, 4]. With prefix tables, a client that is bound to a remote server for file service due to failure of local servers will not automatically switch back to local servers upon successful recovery of local servers. As a result, a client may still rely heavily on a remote server for file service for a long period of time after a crashed local server is back to work again. On the other hand, with clue tables, a client will always turn to local servers first to locate a file when it starts a new file session, i.e. opens a file. This guarantees that the bindings among clients and servers occur in conformity with network hierarchy design whenever possible.

2. It implicitly carries out load balancing among servers to improve system scalability –
   The clue tables mechanism implements dynamic binding between clients and servers upon file open to achieve load balancing among servers. This is the main distinction of the clue tables mechanism when compared with other distributed file systems such as AFS[1, 5], Coda[6], Locus[7], V kernel[8], Amoeba[9], Ficus[10], and Deceit[11] that also implement dynamic binding. With clue tables, a client, upon opening a file, multicasts access requests to servers that have a replica of the file. If more than one server acknowledges the request, the client always chooses the server that responds fastest for binding. With this practice, the clue tables mechanism implicitly carries out load balancing among servers since a server with a lighter load has a better chance to respond faster than a server with a heavier load. Though it is not guaranteed that the server with the lightest load among multiple candidates always responds fastest, a near-optimum load balancing situation should persist most of time.

3. It supports file replication and dynamically designates a primary file copy to resolve possible data inconsistency –
   One crucial issue with file replication is how to maintain data consistency. The clue tables mechanism resolves this issue by dynamically designating a primary file copy. It is the dynamic nature and granularity of the mechanism that distinguishes the clue tables mechanism from other distributed file systems that also employ a primary copy based approach, e.g. Locus[7]. The main reason behind employing the dynamic approach with file-level granularity is to distribute servers' load. With clue tables, when one or more clients attempt to write to a file, one server is dynamically designated as the primary server and all accesses to a file are temporarily forwarded to the primary server. Once the situation that could cause data inconsistency no longer exists, the primary server broadcasts the new version of the file to other servers.

In the following part of the paper, section 2 describes the basic structure of clue tables. Section 3 elaborates on the system operations with clue tables. Section 4 addresses the implementation and performance issues. Section 5 concludes this paper.

2 Basic Structure of Clue Tables

The clue tables mechanism implements a global naming space. The primitive entities in the clue tables mechanism are file collections termed *domains*. A domain is a subtree in the integrated file system of a distributed system. A file server can contain one or more domains while a domain cannot spread over multiple file servers. A domain must be entirely stored on one file server. However, we may have replicated copies of a domain stored in a number of servers. Fig. 1 illustrates the naming architecture with clue tables.

Each file in a domain is an object composed of two attributes: logical file name and physical location. Each file is uniquely identified by its logical file name in the integrated file system. The physical location attribute specifies where the file resides. Clue tables are the directories that the clients refer to for locating a file based on its logical file name. Fig. 2 shows an example of clue tables. A clue table contains a number of entries, each of which corresponds to a domain in the integrated distributed file system.

For example, in Fig. 2, there are two entries. The first entry corresponds to the domain with root "/usr" and the second entry is corresponds to the domain with root "/usr/bin". A clue table entry specifies the file servers that have a copy of the domain. For example, the domain with root "/usr" has three replicated copies on file servers **solar**, **earth**, and **global**.

With file replication, the availability and scalability of the system is significantly enhanced. The multiple servers listed in a clue table entry are grouped and prioritized. For example, in the first entry of Fig. 2, servers **solar** and **earth** form the first group, which is separated from the second group by a semicolon. The

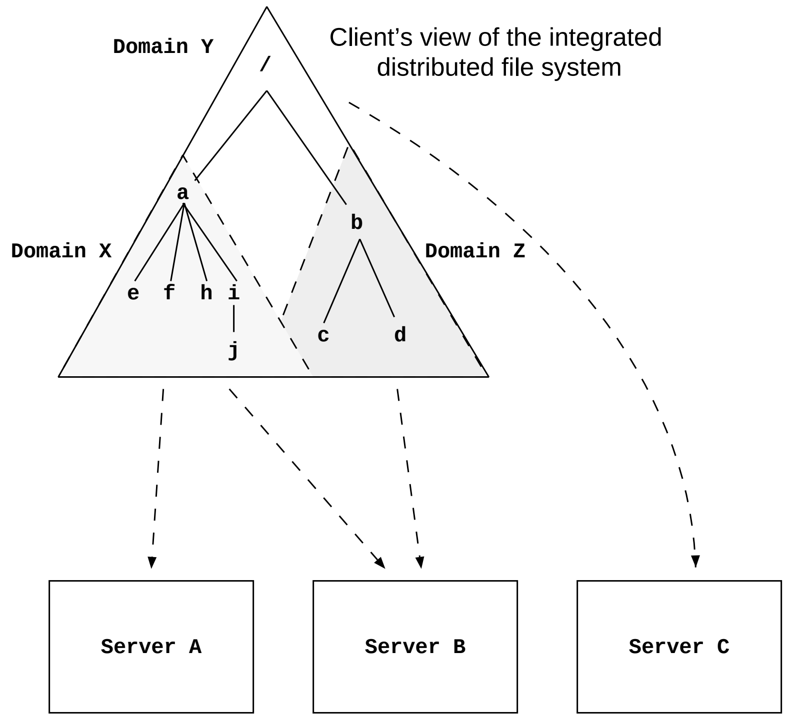


Figure 1: The naming architecture. Domain X is stored on servers A and B. Domain Y is stored on server C. Domain Z is stored on server B.

second group contains only one server, **global**. When trying to locate a file, a client first turns to the servers in the first group. If all the servers in this group are unavailable, e.g. crashed or unreachable due to network failure, the client then turns to the second group and so on. The motivation to group and prioritize servers in the list is to make the bindings among clients and servers occur in conformity with network hierarchy design whenever possible.

A clue table entry is overridden by another entry when the second entry is with root a subdirectory of the first domain. For example, in Fig. 2, the second entry, corresponding to the domain rooted by "/usr/bin", overrides the first entry. When a client tries to locate a file, it first searches the clue table for the longest prefix of the domain that matches the filename. The client then sends requests to the servers in the list.

A clue aliasing mechanism is incorporated to provide more flexibility in system integration. The second entry in Fig. 2 shows an example of clue aliasing. When the client accesses directory "/usr/bin" and sends a request to server **csm** (see Fig. 3), it is actually accessing directory "/usr/sparc/bin" on **csm**. Through clue aliasing, a directory on a server can appear to have different names on different clients. This adds desirable flexibility to system integration.

Certain rules apply to the creation of clue tables in a distributed file system:

1. Each node, a client or server, in a distributed system should have a clue table. A clue table can be shared by two or more nodes but each node must have access to one clue table. The clue table of a node can be stored in a node's local disk if it has one. If the node is diskless, then the clue table is stored in a remote server and is cached by the node in the memory while it is operating.
2. Each node may have some private entries in its own clue table. One good use of this flexibility is the creation of a private "/tmp" directory. If a client has local disks, it may be more appropriate, from the performance aspect, to place temporary files created by this client on its local disks rather than on remote servers. Even if the client does not have a local disk, it may share a localized "/tmp" directory with other diskless clients in the local cluster.
3. If a domain is shared by multiple nodes, all the nodes must contain the same set of servers in their

```
/usr :rep=3:      # three replicated copies.
        solar,    # server name list.
        earth;
        global.   # second group.
/usr/bin :rep=3:  # three replicated copies.
        earth,
        solar,
        csm=/usr/sparc/bin.# clue aliasing.
        ⋮
```

Figure 2: An example of clue tables and clue aliasing.

clue table entries corresponding to this domain. The grouping and prioritizing of these servers may be different, reflecting each node's physical position in the network hierarchy. However, the set of servers in the entries must be identical. Otherwise, data consistency among replicas of the domain cannot be maintained. (Detailed discussion on data consistency guarantees is presented in next section.)

4. When creating a clue table for a node, we should group and prioritize the servers according to their proximity to the node in the network hierarchy. By doing so, the node will always find a file in the nearest available server and its interference with remote nodes in the network will be minimized.

3 System Operations with Clue Tables

This section discusses how the system operates with clue tables. Since the clue tables mechanism implements dynamic binding, the bindings among clients and servers can change on-the-fly. A client establishes a binding to a server upon the start of a file session. A file session is a series of file operations to a file enclosed by open and close operations. When a client starts a file session, it first searches the clue table for the domain that includes the file and sends requests to the servers according to the grouping and priority specified in the clue table. The servers that receive the request respond by locating the file in their own storage and returning a *succeeded* or *failed* message. If replicated copies exist on several servers, the client will choose the server that responds most quickly to a succeeded message for service of this file session. Fig. 3 illustrates the access request flow.

As mentioned earlier, through implementing dynamic binding upon file opening, the clue tables mechanism implicitly carries out load balancing among servers. It is conceivable that a server with a lighter load has a better chance to respond faster than a server with a heavier load. Though it is not guaranteed that the server with the lightest load among multiple candidates always responds fastest, a near-optimum load balancing situation should persist most of time.

When a client selects a particular server for a file session, the client caches the attribute block of the file, the server identification, and a file handle returned by the server to speed up following file operations. The file handle is a unique file index assigned by the server for speedily identifying and locating an opened file. Note that the bindings among clients and servers are per file session basis. A client may turn to different servers for service of different file sessions. Due to network proximity and server load, two clients that are concurrently accessing the same file may be served by two different servers. As a result, access load is distributed over servers and the scalability of the system is significantly upgraded.

The bindings among clients and servers may change during a file session. One reason is to elude access disruption caused by server or network failure. When such a failure occurs, the client will search the clue table for another server that has a replicated copy of the file. If this search succeeds, the client will establish a binding with the second server and the user will not observe service disruption except that the latency of some file operations is longer.

Another occasion in which rebinding is invoked is to maintain data consistency among replicated file copies. With file replication, the system must be able to resolve potential data inconsistency when concurrent writing or read-write sharing occurs.

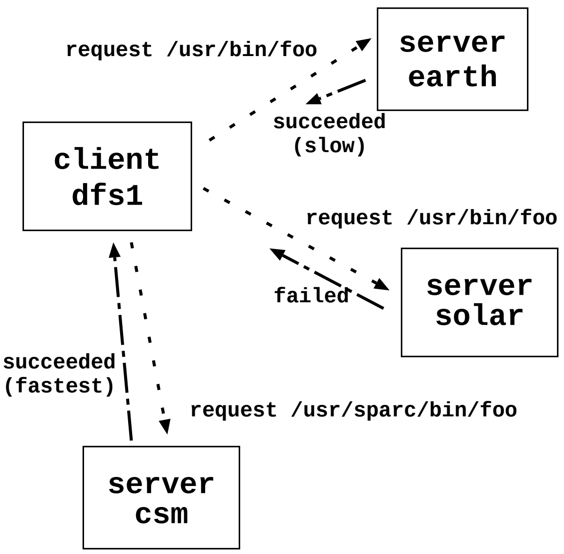


Figure 3: Flow of access requests. In this case, server **csm** is the fastest to reply a *succeeded* message. Client **dfs1** will establish a binding to **csm**'s /usr/sparc/bin/foo.

We addressed the data consistency issue by introducing a primary copy based preventive mechanism. When one or more clients open a file for writing, a server that holds a replica of the file is designated as the primary server. All other servers that also have a replica of the file invalidate their copies. Meanwhile, those clients that are initially bound with the servers that have temporarily invalidated copies will carry out rebinding operations to connect to the primary server. The primary server will provide all accessing services to the file as long as the writing operations continue. Upon termination of the situation, the primary server will broadcast the new version of the file to other servers.

An interesting issue here is how the primary server is selected. As mentioned earlier, the clue tables mechanism dynamically designates the primary server to distribute servers' load. In the situation that only one client attempts to write to the file, the server that receives the write request will become the primary server. If two or more clients want to open the file for writing at the same time, all the servers that receive a write request will compete to become the primary server. A simple arbitration mechanism based on a pre-assigned priority is used to determine the primary server.

4 Implementation and Performance Evaluation

Fig. 4 shows the structure of an implementation of the clue tables mechanism. This implementation is based on Mach 2.6 operating system [12] and the VFS (Virtual File System) [13, 14]. On the client side, the VFS(Virtual File System) forwards file accesses that invoke the clue tables mechanism to the CluFS interface. The CluFS interface checks whether the file access hits the local disk/file cache. If not, the file access request is forwarded to a user-level process called the client daemon. The client daemon looks up the local clue table and multicasts the request to the servers according to the grouping and priority in the matched clue table entry. On the server side, the incoming requests are processed by a user-level process called the server daemon. The server daemon interfaces with the VFS to locate the file and returns a *succeeded* or *failed* message to the requesting client.

To study the performance with the clue tables mechanism, we have conducted two experiments and compared the results with NFS[15]. The experimental system consists of five Intel 80486 based personal computers connected by an ethernet network. All machines run Mach 2.6 operating system and three out of the five act as servers while the remaining two act as clients.

Fig. 5 shows the results from the first experiment. In the first experiment, we repeatedly open and close 30 files to test the overhead of dynamic binding operations. The horizontal axis gives the number of times the operations are repeated. The vertical axis is time the operations take in seconds. Fig. 5 shows that the clue tables mechanism performs slightly better than

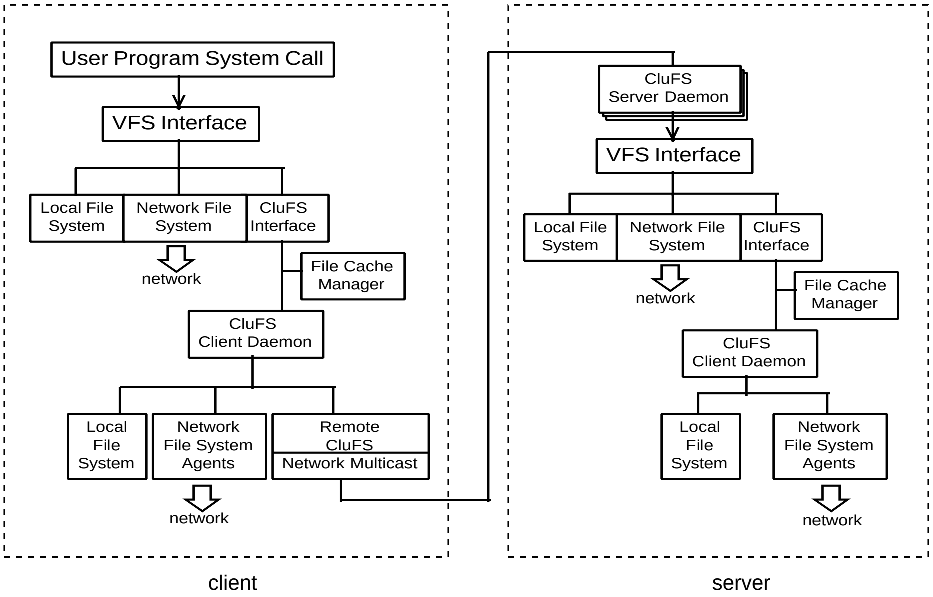


Figure 4: An implementation of the clue tables mechanism.

NFS in file opens. This is due to the use of the socket mechanism[16] in the clue tables file system. The socket mechanism induces less overhead than the RPC (Remote Procedure Call) mechanism used in NFS. For file closes, NFS virtually takes no time. The reason is that NFS uses a stateless file cache coherence protocol and, as a result, does not invoke a remote procedure call when a file is closed.

5 Conclusion

In this paper, we presented a distributed, dynamic naming mechanism called clue tables for building highly scalable, highly available distributed file systems. The clue tables naming mechanism is distinctive in three aspects. First, it is designed to cope well with the hierarchical structure of modern large-scale computer networks. Second, it implicitly carries out load balancing among servers. Third, it supports file replication and dynamically designates a primary copy to resolve possible data inconsistency caused by concurrent accesses to multiple file replicas. The clue tables naming mechanism is incorporated in the Azalea distributed file system currently being developed at National Taiwan University.

Acknowledgments

We wish to thank Mary Baker, the shepherd of this paper, and the reviewers for many helpful suggestions and comments.

References

[1] Mahadev Satyanarayanan. Scalable, Secure, and Highly Available Distributed File Access. *IEEE Computer*, 23(5):9–21, May 1990.

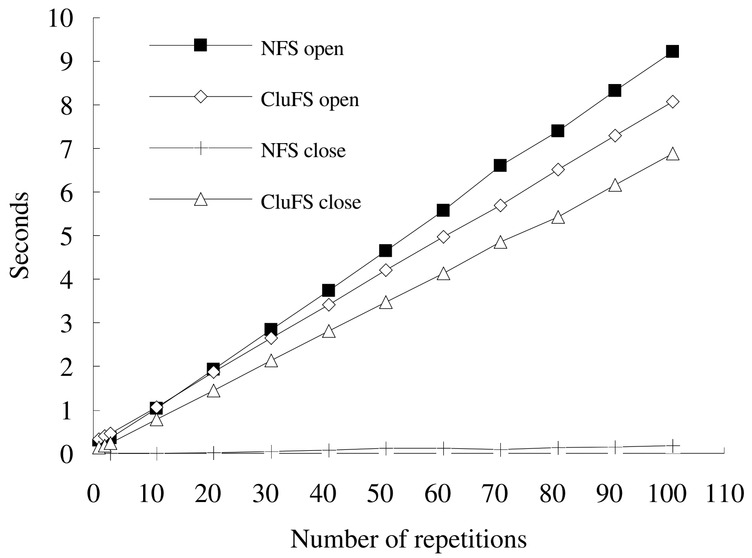
[2] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John Ousterhout. Measurements of a Distributed File System. In *Proc. of the 13th Symposium on Operating Systems Principles*, pages 198–212. ACM, 1991.

[3] Brent Welch and John Ousterhout. Prefix Tables: A Simple Mechanism for Locating Files in a Distributed System. In *The 6th Intl. Conference on Distributed Computing Systems*, pages 184–189. IEEE, May 1986.

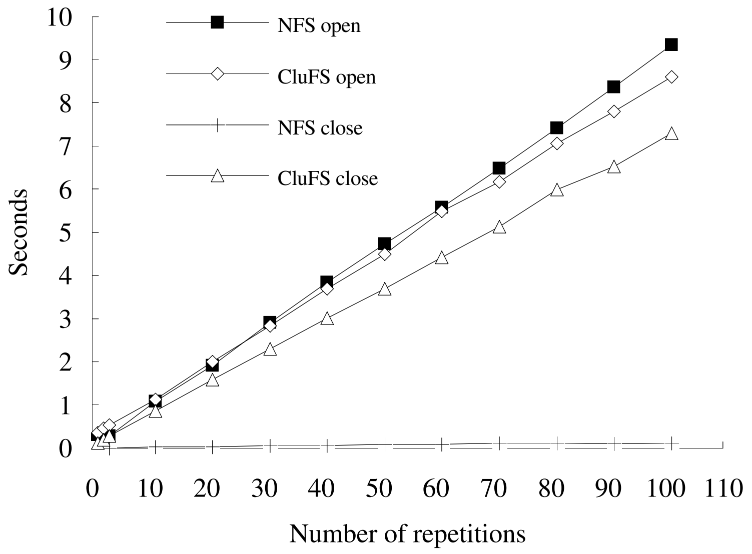
[4] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch. The Sprite Network Operating System. *IEEE Computer*, 21(2):23–36, Feb. 1988.

[5] Mahadev Satyanarayanan, John H. Howard, David A Nichols, Robert N. Sidebotham, Alfred Z. Spector, and Michael J. West. The ITC Distributed File System: Principles and Design. In *Proc. of the 10th Symposium on Operating Systems Principles*, pages 35–50. ACM, Dec. 1985.

[6] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A Highly Available File System for a Distributed Workstation Envi-



(a) Each loop opens and closes 30 read-only files.



(b) Each loop opens and closes 30 write-able files.

Figure 5: 30 different files are repetitively opened and closed.

ronment. *IEEE Tans. on Computer*, 39(4):447–459, Apr. 1990.

[7] Bruce Walker, Gerald Popek, Robert English, Charles Kline, and Greg Thiel. The LOCUS Distributed Operating System. In *Proc. of the 9th Symposium on Operating Systems Principles*, pages 49–70. ACM, Oct. 1983.

[8] David R. Cheriton. The V Distributed System. *Communications of the ACM*, 31(3):314–333, March 1988.

[9] S. J. Mullender and A. S. Tanenbaum. A Distributed File Service Based on Optimistic Concurrency Control. In *Proc. of the 10th Symposium on Operating Systems Principles*, pages 51–62. ACM, Dec. 1985.

[10] Richard G.Guy, John S. Heidemann, Wai Mak, Thomas W. Page, Gerald J. Popek, and Dieter Rothmeier. Implementation of the Ficus Replicated File System. In *USENIX Summer '90*, pages 63–71, 1990.

[11] Alex Siegel, Kenneth Birman, and Keith Marzullo. Deceit: A Flexible Distributed File System. In *USENIX Summer '90*, pages 51–61, 1990.

[12] J. Boykin and A. Langerman. Mach/4.3BSD: A Conservative Approach to Parallelization. *Computer Systems*, 3:69–99, 1990.

[13] Steve R. Kleiman. Vnode: An Architecture for Multiple File System Types in SUN UNIX. In *Summer USENIX Conference proceedings*, pages 238–247, Atlanta, GA, June 1986.

[14] David S. H. Rosenthal. Evolving the Vnode Interface. In *Summer USENIX Conference proceedings*, pages 107–118, Anaheim, CA, 1990.

[15] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network File System. In *USENIX Conference proceedings*, pages 119–130, June 1985.

[16] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quaterman. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley Publishing Company, 1989.

Author Information

Cheng-Zen Yang is currently a Ph.D. student at the Dept. of Computer Science and Information Engineering of National Taiwan University, Taipei, Taiwan. He received the B.S. degree in Computer Engineering from National Chiao Tung University in 1988, and M.S. degree in Computer Science and Information Engineering from the same university in 1990. His major research interests include distributed computing systems and operating systems. He can be reached by e-mail at dennis@solar.csie.ntu.edu.tw.

Chih-Chung Cheng was born in Tainan, Taiwan, R.O.C. He received the B.S. degree in Computer Science and Information Engineering from National Taiwan University in 1992. He is currently in the master program of the same department and expected to complete his degree in 1994. His research interests are distributed file systems and computer networks. He can be reached by e-mail at chung@solar.csie.ntu.edu.tw.

Yen-Jen Oyang received the B.S. degree in Information Engineering from National Taiwan University in 1982, the M.S. degree in Computer Science from the California Institute of Technology in 1984, and the Ph.D. degree in Electrical Engineering from Stanford University in 1988. He is currently an Associate Professor in the Department of Computer Science and Information Engineering, National Taiwan University. His research interests include computer architecture, distributed systems, and VLSI system design. He can be reached by e-mail at yjoyang@csie.ntu.edu.tw.