Distributed File Systems: A Survey

Ying Sun

Supervisor: Dr. Rick Bunt

Abstract

A file system is a key component of any operating system. The purpose of a distributed file system is to allow users of physically distributed computers to share data, storage and administrative support resources by using a common file system. The distribution of users, storage and management services are aspects of file system distribution. Also, the storage and management can be distributed among a group of servers as well as all the machines around the system. This article surveys the design and performance issues of distributed file systems by studying five existing distributed file systems, focusing on the trends and problems needing to be solved. Among the issues considered are the following: naming services; transparency; file caching; availability and scalability, etc. In the future, more attention should be paid to issues such as: heterogeneity, file mobility, effective use of storage subsystems, the change of data type and technical requirement by the introduction of multimedia, and the support for mobile users.

1 Introduction

A file system is a key component of an operating system, whose purpose is to provide long-term storage of data. A distributed file system allows users of physically distributed computers to share data, storage and administrative support resources by using a common file system. In such a system, a client is a process who gives requests, and a server is a process that provides services. Also, the machine on which a client process or server process is running is called a client or a server respectively.

Conventionally, multiple users share files and storage by means of "timesharing". In that model, both storage devices and the file system are located on one central machine, multiple users access the file system in a timesharing mode. In a distributed file system, the notion of sharing doesn't change because logically, it still shows a picture where different users share data and storage resources within one file system. That is why in some of the literature, a distributed file system is viewed as a distributed implementation of the timesharing model ([1],[12]), although the timesharing mechanism is not used there. What makes a difference here is the concept of "distribution". First of all, the multiple users are not confined to one machine. They are dispersed with different computers which are connected by networks (distributed clients). Second, files are stored in the storage of different computers (distributed file servers), and, file management is sometimes done by more than one machine (distributed file management servers). Figure 1 shows the logical and physical structures of file systems. In such an environment, it is challenging to realize the notion of "sharing" in an efficient, secure, and robust manner. This forms the design issues of distributed file systems, including file location, naming schemes, sharing semantics to emulate and remote access method to use. Also, the performance of a distributed file system should be comparable to that of a classic timesharing file system. All of this makes the design and implementation of a distributed file system more complicated.

The future distributed systems will grow larger, more heterogeneous, with more distributed data storage and service. Some issues to which we didn't pay much attention before become more important,

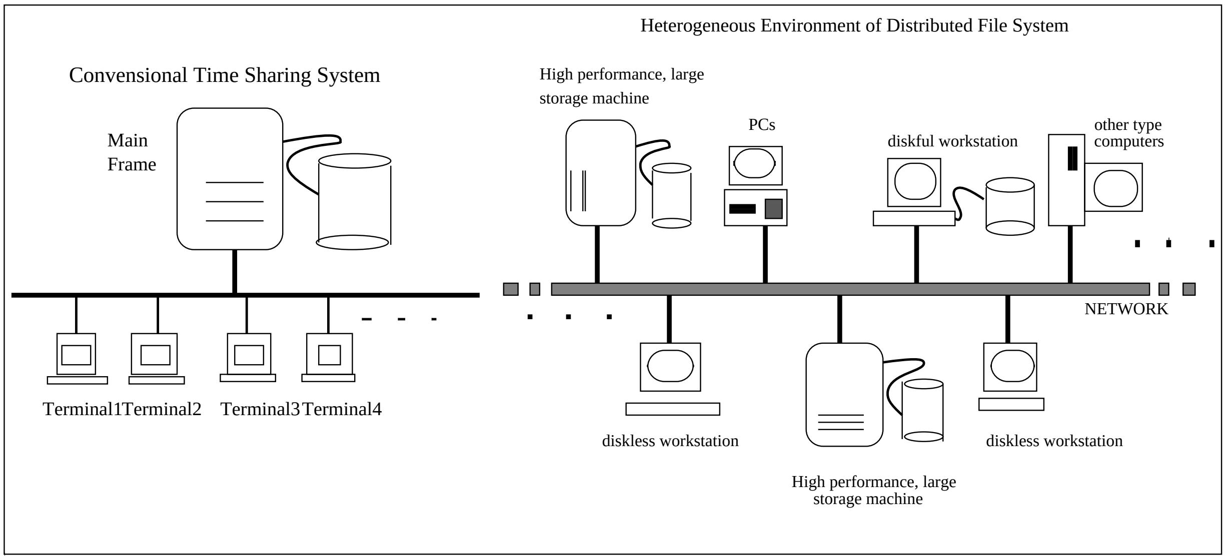

Figure 1: The Structures of Timesharing and Distributed File Systems

eg. the introduction of multimedia, distributed file systems in WAN, and effective use of storage subsystems. Still, goals of distributed file systems such as accommodation to heterogeneity, availability, system scalability and security are far from achieved.

The attempt to build higher performance distributed file systems has never stopped. There are tens of distributed file systems existing. Some recent experimental systems follow a new design (eg. Sprite, [1],[7]), some only make changes on an exist file system (xFS, [8]). The standard NFS ([1],[10],[12]) and AFS ([1],[2],[6],[13]) change the former versions to make new versions with better performance. Also, there are hybrid implementations, which make efforts to bring together the advantages of exist implementations (eg. Spritely NFS, [10],[11]).

In this project, issues and trends of designing distributed file systems are surveyed on the basis of five existing systems. They are: NFS, AFS, Sprite, xFS, and Spritely NFS . The reasons of choosing these file systems are as follows: 1). NFS and AFS have become industry standard. NFS has been widely used for years. AFS has been around for a while, and now becomes important because it was introduced to OSF/DCE as the standard of its DFS. 2). xFS is an experimental file system which makes changes to AFS, to improve scalability, the most obvious advantage of AFS. 3). Sprite is an experimental file system on the basis of using cache extentively, and cache is the commonly used mechanism to improve performance. 4). Spritely NFS is also an experimental file system which provides cache consistency, which NFS fails to do.

In Section 2, we describe the distributed file systems surveyed, and give some overall information about these systems for the discussions in the later sections. Section 3 discusses design issues and related problems. The most interesting performance issues and future trends are included in Section 4. Section 5 is a summary.

2 The Systems Surveyed

In this paper, five different distributed file systems are surveyed. They have different design goals, and follow different approaches. As is evident, the system configuration varies. There are configurations where servers run on dedicated machines, eg. AFS and Sprite, as well as configurations where a workstation can be both a client and a server, eg. xFS. The survey will include the design goal of the system, the mechanism used, and the main achievement.

2.1 NFS: Sun Network File System

NFS is part of the SunOS operating system of SUN Microsystems, Inc. It was introduced in 1985, and has been widely used in industry and academia. Now it runs on various machine types and operating systems: UNIX 4.x BSD, SunOS, DEC Ultrix, System V.x, VMS and MS/DOS etc.

The environment NFS faces is a heterogeneous one, with various machine types and operating systems. The goal is to provide some degree of sharing among the independent file systems in a transparent manner, on the basis of client-server relationship. It is possible for any NFS machine to be a client, a server or both ([12]). Clients mounts subtrees of the file system individually from servers, making a flexible logical structure, without providing a global name space. Clients access data from server by remote service mode, ie, the file operation is done on the server side, and the result is sent back to the client. Cache is only used to improve read performance. A write-through policy is used to handle NFS write operation. NFS server works in a stateless manner (ie. server doesn't keep any information on a client after the request is serviced) and uses the underlying UNIX file protection for security. NFS also provides a service called Yellow Pages (YP) or Network Information Systems (NIS) that provide distribution and central administration of read-only system databases. Even though NFS does not have a good cache consistency policy, the advantage of heterogeneously applicable and reasonable performance still makes it a commercial reality and a de facto standard.

2.2 AFS: Andrew File System

Andrew File System has been under development at Carnegie Mellon University since 1983, and the design has evolved over time, resulting four versions called AFS 1.0(1983), AFS 2.0(1985), AFS 3.0 (commercialized by Transarc Corporation, 1988), and AFS 4.0, which was accepted by the Open Software Foundation as the basis for the data sharing component of its Distributed Computing Environment (DCE) ([10]) in May 1990.

The physical structure of AFS can be described as the following: A small collection of servers jointly provides a shared data repository for a much larger number of workstations. The initial goal of AFS is to build a large scale file system which can support at least 5,000 workstations ([2]). Corresponding techniques such as caching, threading, bulk data transfer, and clustering are used to achieve this goal.

The group of dedicated servers are collectively called Vice. A client process called Venus caches files from Vice, and writes back modifications of files. There is no remote interaction between clients and servers in AFS, and whole file caching or big chunk caching (64Kb, in the later versions above version 3.0) on the client side reduces both client-server data traffic and server load. Another contribution of AFS is volume migration and replication. A volume is AFS's component unit, always formed by the files of a single user. AFS moves volumes among disk partitions and servers to balance the use of storage resources and servers. Read-only replication of volumes containing system executable files and seldom-updated files improve the system availability.

AFS shows that large scale distributed file system is tractable. It also forms the starting point of research on highly-available distributed file system like Coda ([14]), and highly-scalable distributed file system like xFS ([8]).

2.3 xFS: An Experimental File System Focusing On Scalability

Starting from AFS, researchers are trying to move as many server responsibilities as possible to client workstations in order to improve the system scalability and reduce the need of high performance server machines. This results in an experimental file system temporally called xFS, which is under study in the University of California at Berkeley.

The physical structure of xFS is that the storage are distributed among the former client machines (now playing the role of both client and data server),data transfers happen among these machines, only the management is done by the former server (now the management server). Cache write policy is simplified in that no write through is needed; modified files stay at the client machine that modifies them. Cache consistency is based on a mechanism called write ownership. The technique of clustering is inherited from AFS, with the cluster server coordinate the file transfers within a cluster.

The study on xFS protocol shows that it reduces the server load by a factor of six compared to AFS without significantly affecting response time or file availability ([8]). Although xFS requires every client machine have local disk(s) and large memory, and introduce challenges on system administration, backup and security, it represents a new approach of changing client/server to peers, thus still worth studying.

2.4 Sprite Network File System

Sprite is an experimental operating system under development at the University of California at Berkeley, to test on the effective use of large memories, support for multiprocessor workstations, efficient network communication and diskless operation. What we discuss is the file system of it, also called Sprite in this report. Its global file hierarchy is divided into subtrees called domains, each domain has a domain server, which provides data storage and file service within this domain for all the clients around the system. The clients are diskless workstations, with large local memories. File service relies heavily on caching.

In Sprite system, each machine maintains a server map called prefix table, whose function is to map domains to servers. This table is empty initially, and a broadcast protocol adds the needed domain to the table. A build-in facility processes the prefix of whole pathname, and a file lookup operation will result in the longest matching prefix, then only the remainder of the file name will be searched by usual UNIX path name translation. The efficient lookup operation is one of Sprite's contributions.

Based on its physical structure, Sprite makes extensive use of caching. A delayed-write approach is used to handle file modification. Clients check the file version numbers from the server and the cached copy, to decide whether the cached copy is the current version of the file. Server keeps track of the last writer of each file to make sure that it always has the current copy and disables the client cache when concurrent read and write happen. Sprite's caching policy is quite successful in cache consistency, and gain performance in using main memory caching, but in some degree sacrifices performance in using centralized control management of cache and eliminating the possibility and benefits of caching in big chunks. Also, the static division of domains among domain servers constrain the file mobility and system scalability.

2.5 Spritely NFS

NFS is widely used now, but it does not provide correct file access from multiple clients. Spritely NFS is an experiment that grafts Sprite's cache-consistency protocol onto NFS. This experiment is currently being done at Digital Equipment Corporation Western Research Laboratory.

NFS doesn't have client-to-server open and close requests. Spritely NFS introduces them. The server in Spritely NFS maintains a "state table", and checks the table when an open request is issued by a client, whether this open request conflicts with uses by other clients. In case of conflict, a server-to-client "callback" is sent to force the client caching dirty blocks write back the modifications. Moreover, Clients do not have to write-through dirty data on close. Like in Sprite, the server keeps track of each file's last writer. The server requires the client who caches the dirty data to write it back when other clients try to open that file. Otherwise, the last writer will write back the modification at its leisure. This is the "write-behind" policy.

Spritely NFS removes the "stateless server" restriction of NFS, which improves the overall performance. A crash-recovery protocol is being studied and implemented ([11]).

Despite the considerable diversity in the manner in which the above systems are put together, they are all based on a small number of mechanisms, eg. client cache, mounting, and replication. But still, there are some problems to solve in the design of a distributed file system. The following section will discuss some important design issues, and problems that need to pay more attention.

3 Design Issues and Problems

In spite of the difference on physical structures between time sharing and distributed file systems, they show the same picture logically that multiple users access the same file hierarchy simultaneously. To do this, a distributed file system must provide naming service and transparency to some degree. And when two or more users share the same file, it is necessary to define the semantics of reading and writing precisely to avoid problems. The most widely used are UNIX and session semantics ([1],[9]). Caching policies are also important in the design.

3.1 Naming and Logical Structure

One of the primary goals of any distributed file system is to provide naming services that map logical and physical objects. The naming schemes decide the logical structure of a distributed file system. The two commonly used naming schemes that provide transparency are: 1) Local structure: letting individual machines mount remote directories to their local file hierarchy. Each machine can make its data public by exporting, and can use the public data by mounting the subtrees onto its local point. It is a forest, with each machine representing a tree. Because of its physical structure, NFS uses this scheme, which fits well with the heterogeneous environment. It brings the disadvantage that the shared name space may not be identical at all the workstations. 2) Global structure: building a global name structure that spans all the files in the system. Each machine sees a huge tree representing the whole file system, and can mount any of the subtrees to its local point. This is more similar to the logical structure of the conventional timesharing model. AFS, Sprite and xFS, all use the second scheme. The problem is that even if physically, the storage can be distributed among the machines, the file access information should be coordinated in the central server to maintain a consistent global file hierarchy (even in xFS).

The mount mechanism is the most commonly used technique in implementing almost all the current distributed file systems ([1], [3]). Even though used slightly different in the file systems, it is the common method to bring remote file systems to the global file hierarchy or local site.

3.2 Location Transparency and File Mobility

When transparency is discussed in the context of file names, these two notions are always taken into account.

Location transparency means the name of a file does not reveal any hint as to its physical storage location. Most of the current distributed file systems we surveyed, eg. NFS and Sprite, provide location transparency. It provides users with a convenient way to share data, but since the logical names are still statically attached to physical storage devices, the sharing of storage space becomes cumbersome. By contrast, file mobility (also referred to as location independence or file migration), means the name of a file need not be changed when the file's physical storage location changes ([1]). It separates the naming hierarchy from the storage device hierarchy and the inter-server structure, makes the files be viewed as logical data containers not attached to a specific storage location. Because of the dynamic nature, it is a stronger property. It will be useful in the environment with a distributed storage server. In reality, only AFS supports file mobility, and only for administration purpose. File mobility should be an important topic that needs more research, especially when the storage is distributed among many servers or machines in the future systems.

3.3 Caching and Related Problems

Every current distributed file system uses caching techniques to reduce network traffic, delays and contention for the disk arms ([7]). "Cache whenever possible" was a design principle for AFS ([3]) to gain good performance, and Sprite design is based on large client cache ([7]). Traditionally, cache is used to improve read performance, but there are now new studies on write caching([4]).

Studies on cache size give definite results on miss ratio curve, and the well-known phenomenon of "diminishing returns" ([4]). Another problem concerning cache size is the balancing of client and server cache sizes when a system uses caches on both client and server side, because increasing the size of client cache may also increase the server miss ratio. Sprite gives a solution on this by using dynamic size client cache ([7]).

The granularity of the cached data (also called caching unit size) can vary from parts of a file to an entire file. Increasing the caching unit size increases the hit ratio and reduces network overhead, but increases the time required for data transfer and the potential for consistency. In the surveyed file systems, AFS uses entire file caching in early versions, and performs caching in big chunks(64Kb) now ([2]). NFS uses an 8Kb cache unit, and Sprite uses only 4Kb.

Cache unit size is tightly related to the data transfer unit. Data transfer overhead between client and server can be reduced using bulk data transfer, by amortizing fixed protocol overheads over many consecutive pages of a file. AFS transfers entire files or large chunks of files ([3]). NFS and Sprite also exploit bulk data transfer by using very large packet size (often 8Kb). The workload of the system should have new characteristics when using bulk data transfer, and this will need further research.

Cache modification policies are also called write policies. The commonly used write policies include: write through, delayed write and centralized control policy. Write through means clients write back everything to the server when write happens. Delayed write is divided into three schemes: write-before-ejected (from the cache), write-periodically, and write-on-close. Centralized control requires the server to keep track of the opened files, so that when a file is being written, all other accesses are prevented. NFS uses the write through policy. Sprite uses periodic write policy that writes back all the changes in a period of 30 seconds. xFS partially used central control. AFS uses on-close write as its write policy. There is a tight relationship between write policy and sharing semantics. A summary of cache modification policies is given in Table 1. Work being done on write policies should lead to more complicated or hybrid ones. Write-behind, which used in Spritely NFS, is an example.

If more than two clients cache the same file, and one of them modified it, all the other copies should be invalid to keep the consistency of the data. This is called cache validation. It has two approaches: client-initiated approach and server-initiated approach. The first approach often causes network traffic, this makes some of the distributed file system designers withdraw from the client-initiated approach. In the other approach, the degree of server involvement depends on the sharing semantics the file system intends to emulate. Server-initiated approach violates the traditional client-server model, where clients initiate activities by requesting services.

Table 1: Summary of Cache Modification Polices

	Advantages	Disadvantages	Sharing Semantics
Write Through	Reliability	Can't exploit cache by writing	Hard to decide
Write Before Ejected	Better Performance	Some blocks can reside long time before written	Ambiguous
Periodical Write	Performance, Prevent long reside	Reliability Problems	Combined with other schemes can emulate UNIX semantics
Write On Close	Reduce Net traffic Only in cases frequently modified or opened long period	Make the closing process to delay	Exact session semantics
Centralized Control	Consistency	Not robust Scale poorly	UNIX semantics

3.4 Accommodating Heterogeneity

As a distributed system grows. it tends to grow more heterogeneous. The system heterogeneity arises from the diverse users whose different requirement are most naturally met by particular kinds of systems. Different types of heterogeneity includes the heterogeneity of language, file system, operating system, and hardware platform. A variety of distributed file systems accommodate heterogeneity to some degree. PC-NFS allows PC-DOS applications to access files on an NFS server; PCserver enables PC-DOS applications to access file in AFS Vice ([12]). Also, efforts are made to build a Heterogeneous file system, (eg. HFS [15]) which is considerably different from common distributed file systems. Work still needs to be done on the organization of files in such a file system, and further consider the performance issues such as reliability, availability, locking and transactions ([15]).

Heterogeneity is a chronic problem, and will not disappear in the future. Instead, future systems tends to be more heterogeneous because of the presence of multiple computing environments. An evolutionary approach is to build a distributed computing environment (DCE) on top of existing operating systems. It provides a platform for running distributed applications. This approach hides heterogeneity behind a homogeneous platform. AFS is used as the wide area part of DCE's distributed file system (DFS).

4 Performance Challenges and Future Trends

Now distributed file systems are widely used in industry and academia. A single remote failure may seriously inconvenience a large number of users. This makes the problem of availability acute. Users of portable workstations disconnect from the system for a short time, or the clients when server just recover from crashes, face the problem of consistency. The workload of a distributed file system is also increasing, cause the system to scale. As the performance of other parts of the system improves, the storage subsystem becomes the bottleneck of system speed. These are the main problems we are to discuss in this section.

4.1 Remote Failure and Disconnection

As described in [1], a file is called available if it can be accessed whenever needed, despite machine and storage device crashes. There is a growing need for distributed file systems that are resilient to failures. Redundancy techniques are used to implement availability. Replication of files is a useful redundancy. The replications are the multiple copies of selected files. There are two approaches: replication of files on separate file servers, and replication of files on different media on the same machine (eg. mirrored disks). The most considered approach is to replicate files on different servers, for the following reasons:

- in case one server goes down, no file is lost
- in case one server goes down, file access can still occur
- split the workload over multiple servers

The basic issue on replication is: on the server side, the different replica of the same file reside on failure-independent machines; on the user side, transparency. The main problem associated with replications is that complicated update propagation protocol should be used to provide consistency. Whether we will sacrifice consistency (under some circumstance of catastrophic failure) for availability and performance is another problem to consider. One approach on which consistency can be traded for performance is the replication of the location hints. A hint is a piece of information that can improve performance when correct, but has no semantically negative consequence if erroneous. Since hints are validate upon use, there's no need to maintain their replication consistency. The popular compromise is read-only replication (NFS and AFS). Other kind of replications include data level read-write replication and directory-level replication.

Disconnected operation is another performance-related problem that concerns remote failure, as well as mobile users who have to disconnect from the system for a short period. In providing disconnected operation, theoretically one has to preload the cache with critical data, continue normal operation until disconnection, log all changes made while disconnected and replay them upon reconnection. This requires major modifications to cache management policies because some of the current strategies are not viable when supporting disconnected operation as mentioned in [14]. Coda and Cedar, two systems that are not included in our survey, support disconnected operation to some degree, they proved that disconnected operation is feasible, efficient and usable, but left works to do and research in details in the future.

4.2 Further Scaling

The capability of a system to adapt to increased service load is called scalability. Adding resources may help to deal with the increased workload, but it may also generate indirect load on other system resources (eg. network, or management server), that finally adding of resources can't help. A scalable design should withstand high service load, accommodate growth of the user community, and enable simple integration of added resources.

In considering large scale distributed file systems, the system structure will be an important problem. The former study on very large-scale systems tells us that the service demand from any component of the system should be bounded by a constant. This constant is independent of the number of nodes in the system ([1]). This is called Bounded Resources Principle. This tells us that central control schemes and central resources should not be used to build scalable systems. NFS has a flexible logical structure which can make it easier to scale, but most NFS systems are based on central file servers, which limit its scalability. Sprite is not a scalable design because of its static group of domain servers. AFS makes a progress on scalability in dividing its clients and servers into clusters. The storage of data is dispersed among cluster servers, and within cluster, the system complies with the Bounded Resources Principle. AFS now can support up to 5,000 workstations that have access to a uniform file name space. Basing on AFS, xFS tries to distribute the data storage and file service among clients, which makes a peer-to-peer system with several cluster management servers and a central management server. By using client-to-client data transfer and reducing client-server data stream to that of management information, xFS improves the system scalability by a factor of large ([8]).

Distribution of storage and file service among former clients wins scalability, but leaves problem on system administration. NFS's YP service may give an example of solving such a problem. Up to now, despite the distribution of users, storage, and file service, we still have the centralized coordinator of file access information. Dispersion of the will make the system more distributed, and may improve the scalability, but causes the problem of not maintaining the entire file system as a whole.

4.3 Other Trends

The speed to satisfy service requests is an important goal of distributed file systems. Attentions have been paid to the overhead of network transfer and the CPU overhead to run the communication protocol software. But there is another important overhead of disk access. Compared to the improvement of performance on other parts of the system, the storage subsystem becomes the bottleneck of the system performance. More efforts need to be made on making effective use of storage subsystem.

There are two approaches to improve disk performance. One is to change the file system design to speed up writing and make full use of disk bandwidth. The example is log-structured-file system (LFS, [17]). This file system has an only structure in the disk called "log", and writes all new information to disk in a sequence of logs in a single disk write operation. [17] proved that LFS successfully improved I/O performance. The most difficult challenge is large free areas maintenance, which is still under study.

The other approach is new disk techniques like RAID (Redundant Array of Independent Disks, [16]). It can be mainly described as follows: Data are striped in disk unit arrays distributed across disks. This allows parallel accessing to multiple disks, increases I/O bandwidth, and provides possible redundancy of data which improves availability. This is based on multiple disks of a single machine. In the implementation of xFS, this technique is used on client side. More study and experiment are needed.

Another challenge is brought by the introduction of multimedia applications. File size can be much larger than that of a text file, eg. video files often exceed a gigabyte. Text service seldom requires real-time service, but video does. To handle such applications, we may need a completely different file system than the ones we have now.

Future trends also include distributed file systems on wide area networks (WAN), fault tolerance ..., a lot of challenges await the designers.

5 Summary

In studying distributed file systems, we notice that the physical structures are different, and this leads to different techniques used in building them. The storage, authentication, and naming services are all no longer to be reside at a central server. They at least should be distributed among a group of servers, or even among clients. That means the system is becoming more and more distributed. At the same time, the system remains heterogeneous. New and better hardwares have already improved the performance of individual machines around the system. How to make good use of these improvements to increase the overall performance of the system is also the task of the designers of a distributed file system.

In designing the current distributed file systems, mechanisms like caching, replication, mount points, and bulk data transfer are commonly used. Naming and transparency, as well as sharing semantics are important design issues. File mobility, further accommodation of system heterogeneity, and workload characteristics when using bulk data transfer are problems need more study.

Mobile users of the system is a newly emerging challenge for system designers. Disconnected operation was proven to be feasible in solving problems in remote failure. It may also be applied to the problem of mobile users, but detailed studies are needed in both areas.

Scalability will remain an important performance issue. It will concern not only the increasing of workload, but also the coverage of wide area of networks. Clustering should be useful and needs more study.

Multimedia applications present a new challenge for distributed file systems. The characteristics of data and the file service requirement as well as the system behaviour will all be different from that of current systems. A new design may be needed for a multimedia server.

The solutions for problems concerning distributed file systems can be provided not only by means of file system technology, they also inspire issues in other areas of computer technology, eg. I/O techniques, networking. Many exciting challenges await the designers of future distributed file systems.

References

[1] Levy, E. and Silberschatz, A. Distributed File Systems: Concepts and Examples. *ACM Computing Surveys 22*, 4 (Dec. 1990), 321-374.

[2] Howard, J., Kazar, L. M., Menees, S. G., Nichos, D. A., Satyanarayanan, M., Sidebotham, R. N., and West, M. J. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems 6*, 1 (Feb. 1988), 51-81.

[3] Satyanarayanan, M. Scalable, Secure, and Highly Available Distributed File Access. *IEEE Computer 23*, 5 (May 1990), 9-21.

[4] Chen, K., Bunt, R. B., and Eager, D. L. *Write Caching in Distributed File Systems.* Master's Thesis, Department of Computational Science, University of Saskatchewan, Aug. 1994.

[5] Renaud, P. E. *Introduction to Client/Server Systems.* Wiley, 1993.

[6] Stolarchuk, M. T. Faster AFS. In *1993 Winter Usenix*, San Diego, CA, Jan. 1993, pp.67-75.

[7] Nelson, M. N., Welch, B. B. and Ousterhout, J. K. Caching in the Sprite Network File System. *ACM Transaction on Computer Systems 6*, 1 (Feb. 1988), 134-154.

[8] Dahlin, M. D., Mather, C. J., Wang, R. Y., Anderson, T. E., and Patterson, D. A. A Quantitative Analysis of Cache Policies for Scalable Network File Systems. *Performance Evaluation Review 22*, 1 (May 1994), 150-160.

[9] Tanenbaum, A. S. *Distributed Operating Systems,* Prentice-Hall, 1995.

[10] Borghoff, O. F. *Catalogue of Distributed File/Operating Systems.* Springer-Verlag, 1992.

[11] Mogul, J. C. Recovery in Spritely NFS. *Computing Systems 7*, 2 (Spring 1994), 201-262.

[12] Satyanarayanan, M. A Survey of Distributed File Systems. *Annual Review of Computer Science 4*, 1990, 73-104.

[13] Lever, C. Using DFS Without DCE/LFS. Tech. Rep. 94-2, CITI, U. Michigan.

[14] Kistler, J. J. and Satyanarayanan, M. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems 10*, 1 (Feb 1992), 3-25.

[15] Pinkerton, B. C., Lazowska, E. D., Notkin, D., and Zahorjan, J. A Heterogeneous Distributed File System. In *Proceedings of the 10th International Conference on Distributed Computing Systems,* Paris, France, 1990, pp.424-431.

[16] Ganger, C. R., Worthington, B. L., Hou, R. Y., and Paff, Y. N. Disk Arrays, High-Performance, High-Reliability Storage Subsystems. *IEEE Computer 27*, 3 (Mar. 1994), 30-36.

[17] Rosenblum, M. and Ousterhout, J. K. Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems 10*, 1 (Feb. 1992), 26-52.