A Comparison of Two Network-Based File Servers

(Summary)

James G. Mitchell
Xerox Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304

Jeremy Dion
Cambridge University Computer Laboratory
Corn Exchange Street, Cambridge, UK CB2 3QG

1. Introduction

This paper compares two working file servers in terms of their design goals, implementation issues, performance, and service experience. One server, the Xerox Distributed File System (XDFS) [10], was built at the Xerox Palo Alto Research Center; the other, the Cambridge File Server (CFS) [2, 3, 4], was built at the Cambridge University Computer Laboratory.

Both file servers support concurrent random access to files over a network, and each offers an atomic transaction mechanism covering modifications to files.

2. Underlying Hardware and Software

The XDFS was written in Mesa [6], runs on an Alto minicomputer [11], and communicates with its clients using an Ethernet-1 communications system [5]. The CFS was written in BCPL [9], runs on a Computer Automation LSI4/30 minicomputer, and uses the Cambridge Ring [12] as the communication medium. The servers use similar disk units.

3. General Design Choices

The CFS was originally conceived as a replacement for the backing store management of the CAP computer at the Cambridge University Computing Laboratory. It was required to provide rapid access in a style suited to a virtual memory system which would swap entire segments across the network. This concentration on serving operating system clients efficiently has led to restrictions in the interest of simplicity. For example, the CFS locks entire files at a time with multiple-reader-single-writer interlocks for the duration of a transaction and allows only one file to be updated atomically in a single transaction.

The XDFS was intended to provide a basis for data base research. An XDFS transaction can cover updates to a number of files so that the atomicity of a higher level database operation can be maintained.

The CFS provides access control based on *capabilities*. The XDFS uses identity-based access control on the *names* of files.

An unstructured set of files identified only by UIDS provides the basic interface to each file server, but leads to the possibility for lost and undeletable files, since clients must be trusted to delete files explicitly. The XDFS provides only a partial solution to this problem while the CFS's structures enable it to use garbage collection to delete files that are no longer accessible by clients.

The differences between the CFS's and the XDFS's network protocols stem almost totally from the following differences in the servers:

- the CFS assumes that transmission is over a local, extremely reliable network only, in contrast with the XDFS, which assumes that transmission is over a non-local, mostly reliable internetwork.
- the XDFS is prepared to process more than one request at a time, whereas the CFS handles exactly one request at a time, which makes resource utilization more predictable.

4. Comparison of Implementations

In the CFS, the information needed to map a (file identifier, page number) pair into the disk address of the page is recorded in a tree of disk pages per file. The upper levels of these trees are disk pages containing arrays of disk addresses, and only the leaf pages contain the actual data of the file or index.

The XDFS uses a B-tree [1] to translate a (file identifier, page number) pair into a disk address. There is a single B-tree per disk pack, and it records the allocations of pages to files for all files on the pack.

Though the two servers use quite different representations for files, both obey two rules when performing disk writes so that the consistency of disk information can be maintained across failures:

(1) Any page which will be returned to the free pool in the event of a failure can be written without precautions. All data pages of client files fall into this category; rather than overwriting the current page of a file with changed data, both the CFS and the XDFS allocate a new *shadow* page for the data and record the fact that a modification to the file accessing structures will be needed if the transaction completes successfully. In the event of a failure, these shadow pages are released.

(2) All other pages must be redundant at the time of writing: either their old contents or their intended contents must be reconstructible only by examining other disk blocks. In this category fall all the server's structural pages: the XDFS B-tree and free page map, and the CFS object trees and cylinder maps to be described below.

The XDFS provides this structural redundancy using an abstraction called *stable storage* [10]. Pages of essential information, such as those of the B-tree or the page-allocation bit map, are recorded redundantly on disk. A write to a page of stable storage must be done carefully: writing on the second page must not start until it has been verified that the first has been written successfully. It is assumed that a crash while the write heads are actually turned on will leave a page *detectably bad*; i.e., future attempts to read it will fail because of CRC or ECC errors.

In the CFS, this structural redundancy is built into the disk representation at a higher level. On each disk, one block per cylinder is reserved as a *cylinder map*, an array of entries indexed by sector number, with one entry for each disk page on the cylinder. Each entry contains both the allocation state of the page, and, if it is currently in use, the UID of the object to which it belongs and the tree address it occupies within that object.

Cylinder maps and the pointer pages of object trees are mutually redundant: The current use of each disk page is described both by its cylinder map entry and by its presence in an object tree. As long as the server is careful to write a cylinder map only when the set of object maps is consistent and an object map only when the set of cylinder maps is consistent, the redundancy of each disk page of structural information is maintained.

6. Suggestions for Future File Servers

There is one obvious area for investigation in the design of future file servers. The requirement to survive the failure of a disk transfer has exacted a substantial cost in complexity and in actual transfers per transaction in both the CFS and the XDFS. Maintaining this level of robustness requires a doubling of the number of disk writes to structural pages.

An attractive solution to this problem is to make some of the server's memory non-volatile, so that it will survive all arbitrary halts by the processor, even a power failure during a disk write [7].

Both the CFS and the XDFS present a "flat" view of transactions, in that a transaction cannot contain any others. Reed has proposed a scheme which allows nested transactions [8]. In many circumstances, the simple steps in a transaction will normally cause the entire transaction to fail in turn. Nested transactions may turn out to be advantageous, however, if the failure of a particular subtransaction can be dealt with by trying an alternative subtransaction which might lead to eventual success. More experience is needed to decide which of these two views is most generally appropriate.

References

[1] R. Bayer and E.M. McCreight, "Organization and Maintenance of Large Ordered Indexes", *Acta Informatica*, 1, pp. 173-189, 1972.

[2] A.D. Birrell and R.M. Needham, "A Universal File Server", *IEEE Trans Software Eng*, SE-6(5) pp. 450-453, Sept. 1980.

[3] J. Dion, "The Cambridge File Server", *Op Sys Rev*, **14(4)**, pp. 26-35, Oct. 1980.

[4] J. Dion, "Reliable Storage in a Local Network", Ph.D. Dissertation, Cambridge University, February 1981.

[5] R.M. Metcalfe and D.R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks", *CACM*, 19(7), pp. 395-404, July 1976.

[6] J.G. Mitchell, W. Maybury, and R.E. Sweet, "Mesa Language Manual", *Report CSL-79-3 Xerox PARC, Palo Alto, CA*, April 1979.

[7] R.M. Needham, J.G. Mitchell, and A.J. Herbert, "How to Connect Stable Memory to a Computer", *Op Sys Rev to appear*.

[8] D.P. Reed, "Naming and Synchronization in A Decentralized Computer System", Ph.D. Dissertation, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, Sept. 1978. Also available as MIT Laboratory for Computer Science Technical Report TR-205, Sept. 1978.

[9] M. Richards, "BCPL: A Tool for Compiler Writing and System Programming", *AFIPS SJCC Conference Proceedings*, **35**, pp. 557-566, 1969.

[10] H.E. Sturgis, J.G. Mitchell, and J. Israel, "Issues in the Design and Use of a Distributed File System", *Op Sys Rev*, 14(3), pp. 55-69, July 1980.

[11] C.P. Thacker, E.M. McCreight, B.W. Lampson, R.F. Sproull, and D.R. Boggs, "Alto: A Personal Computer", *Report CSL-79-11 Xerox PARC, Palo Alto, CA*, Aug. 1979.

[12] M.V. Wilkes and D.J. Wheeler, "The Cambridge Digital Communications Ring", *Proc Local Area Communications Network Symp*, Boston, May 1979, Nat. Bur. Standards Special Publication.