Autonomy or Independence in Distributed Systems?

Position Paper

M. Satyanarayanan
Department of Computer Science
Carnegie Mellon University

The topic of this workshop can best be characterized by the question ''How does one obtain the benefits of sharing without the negative consequences of dependence?'' An important first step in addressing this question is to ask what is being shared and why. The sites of a distributed system share two kinds of entities, *resources* and *data*.

The use of remote resources such as printers, specialized hardware, execution cycles, disk storage and so on is motivated primarily by economic considerations. It is often cheaper and more convenient for a collection of sites to share in the use of a common pool of such resources. If the negative consequences of sharing rise to intolerable levels, they can alleviated by adding local resources. For example, one can add a local disk or printer to a workstation, thereby reducing its dependence on the rest of the distributed system for computational resources. It is also often the case that an application is indifferent to the *specific* remote resource it is using. For example, one remote Vax is as good as another identical one as far as execution cycles are concerned. Thus it is possible to develop strategies to replace loss of remote resources by other equivalent resources. The challenge, of course, is to come up with mechanisms (such as process migration, in the above example) that make such alternative use of resources convenient, efficient, and transparent to applications.

The sharing of data, on the other hand, poses deeper and more fundamental problems. Each site in a distributed system is a potential creator and modifier of data. Data sharing arises because a site may wish to make its own actions contingent upon the current value of data at a remote site. Distributed file systems and distributed databases are the two mechanisms that have been evolved to support this. The difference between them is primarily one of focus.

File systems optimize access for small (typically less than 1 Mbyte) data objects that are read or written in their entirety. Most importantly, such systems are predicated on concurrent write-sharing being rare. Studies [7, 10, 8, 5, 1] of file systems have confirmed the robustness of these assumptions over a wide variety of operating systems and usage environments[1]. In contrast, databases focus on applications where fine-granularity concurrent write-sharing by multiple users is the norm. Viewed in this framework, it should be evident that file systems are inherently easier to distribute than databases.

Many distributed file systems such as Sun NFS [11], Amoeba [3], AT&T RFS [6], Sprite [4], and Andrew [9, 2] have been built in the last few years. They have addressed, in varying degrees, issues of location transparency, preservation of local semantics, scalability and security. All of them are based on the client-server model, with the server being treated as the repository of the most current copy of a file. Although some of these systems use caching to improve performance, all of them require the server to be accessible in order to use the files stored on it.

As dependence on distributed file systems increases, availability becomes a serious concern. How can sites in a large distributed environment share data without being critically dependent on each other or on the network? Although motivated by considerations of scalability and performance rather than availability, the Andrew File System took steps in the right direction to address this problem. The caching of entire files on local disks allows most read accesses to avoid server interactions entirely. Write accesses, however, do require a file to be written to its server upon a close operation. To avoid inconsistencies, the Andrew File System does not allow writing of a cached file when its server or the network is down. Thus Andrew File System is sensitive to failures of servers and the network even though it makes minimal demands on them when when they are up.

---

[1]It is, of course, possible that applications have evolved to possess these properties because file systems support this mode of access best. I know of no scientific way to resolve this chicken-or-egg problem. It matters little, anyway, for the task at hand.

How can one build a distributed file system that retains the scalability and security properties of Andrew but is resilient to server and network failures? This question is being addressed in my current work on the Coda File System. The design of Coda uses two distinct but complementary approaches to provide high availability. First, replication is used to guard against server crashes and network partitions. Second, an inconsistency detection mechanism and an exception handling mechanism are used to allow read-write access to data under a wide variety of failure conditions. Coda uses parallel RPC to minimize the performance overheads of replication.

Availability, *performance* and *consistency* are three fundamental attributes of a distributed file system. Unfortunately, these attributes cannot be optimized independently of each other. For example, one cannot allow data to be updated by all sites of a partitioned network if inconsistent updates are intolerable. Coda's goals are to provide maximum availability with minimal loss of performance. Within the constraints of these goals, it will provide the highest level of consistency. Coda takes the view that inconsistency is tolerable in a distributed file system provided that it is rare and detectable, and provided there are mechanisms to help users resolve inconsistencies. This position, first advocated by Locus [12], seems viable primarily because of the relative infrequency of write-sharing in a distributed file system. Experience with Coda will reveal whether this approach is valid in a real system and if the expected gains in availability are indeed realized.

References

[1] Floyd, R.
Short-Term File Reference Patterns in a Unix Environment.
Technical Report TR-177, Department of Computer Science, University of Rochester, March, 1986.

[2] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, 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), February, 1988.

[3] Mullender, S.J. and Tannenbaum, A.
A Distributed File Service Based on Optimistic Concurrency Control.
In *Proceedings of the Tenth ACM Symposium on Operating System Principles, Orcas Island*. December, 1985.

[4] Nelson, M.N., Welch, B.B., and Ousterhout, J.K.
Caching in the Sprite Network File System.
ACM Transactions on Computer Systems 6(1), February, 1988.

[5] Ousterhout, J.K., Da Costa, H., Harrison, D., Kunze, J.A., Kupfer, M., and Thompson, J.G.
A Trace-Driven Analysis of the Unix 4.2BSD File System.
In *Proceedings of the Tenth ACM Symposium on Operating System Principles, Orcas Island*. December, 1985.

[6] Rifkin, A.P., Forbes, M.P., Hamilton, R.L., Sabrio, M., Shah, S. and Yueh, K.
RFS architectural overview.
In *Usenix Conference Proceedings, Atlanta*. Summer, 1986.

[7] Satyanarayanan, M.
A Study of File Sizes and Functional Lifetimes.
In *Proceedings of the Eighth ACM Symposium on Operating System Principles, Asilomar*. December, 1981.

[8] Satyanarayanan, M.
A Synthetic Driver for File System Simulations.
In *Proceedings of the International Conference on Modelling Techniques and Tools for Performance Analysis, Paris,*. May, 1984.

[9] Satyanarayanan, M., Howard, J.H., Nichols, D.A., Sidebotham, R.N., Spector, A.Z., and West, M.J.
The ITC Distributed File System: Principles and Design.
In *Proceedings of the Tenth ACM Symposium on Operating System Principles, Orcas Island*. December, 1985.

[10] Smith, A.J.
Long Term File Reference Patterns and their Applications to File Migration Algorithms.
IEEE Transactions on Software Engineering 4(7), July, 1981.

[11] *Networking on the SUN Workstation*
Sun Microsystems, Mountain View, CA, 1986.

[12] Walker, B., Popek, G., English, R., Kline, C., and Thiel, G.
The LOCUS distributed operating system.
In *Proceedings of the Ninth ACM Symposium on Operating System Principles, Bretton Woods*. October, 1983.