Availability and Consistency Tradeoffs in the Echo Distributed File System

Andy Hisgen, Andrew Birrell, Timothy Mann,
Michael Schroeder, and Garret Swart
DEC Systems Research Center

July 1, 1989

Abstract

Workstations typically depend on remote servers accessed over a network for such services as mail, printing, storing files, booting, and time. The availability of these remote services has a major impact on the usability of the workstation. Availability can be increased by replicating the servers. In the Echo distributed file system at DEC SRC, two different replication techniques are employed, one at the upper levels of our hierarchical name space, the name service, and another at the lower levels of the name space, the file volume service. The two replication techniques provide different guarantees of consistency between their replicas and, therefore, different levels of availability. Echo also caches data from the name service and file volume service in client machines (e.g., workstations), with the cache for each service having its own cache consistency guarantee that mimics the guarantee on the consistency of the replicas for that service. The replication and caching consistency guarantees provided by each service are appropriate for its intended use.

Introduction

Echo is a new distributed file system being designed and built at DEC SRC, with the primary goals of exploring issues of scaling, availability, and performance. Echo provides a global, hierarchical name space, for scaling and for uniformity of access. Replication is employed for availability. Performance is achieved by distributed caching on clients, and by using a log on the file server to reduce disk seeks [3].

89TH0281-6/89/0000/0049$01.00 © 1989 IEEE

Our goal for availability is to tolerate a single failure of a server and keep providing service. Our failure model is that servers are fail-stop, but clients can be Byzantine [6]. With Echo, Byzantine clients can cause denial of service by generating load, but cannot cause corruption of data or of other clients. These assumptions seem reasonable for a workstation environment, where server machines can be placed in computer rooms but workstations will often be less physically secure. We assume that the network can partition and can lose or delay messages.

In this paper, we concentrate on the issue of tradeoffs between consistency of replication and caching versus availability. The file volume service is more consistent but less available, and the number of replicas that it is suited to is relatively small, in the range of, say, one to four. The global name service is less consistent but more available, and the number of replicas that it can accommodate is relatively larger, say, up to several dozen. A survey of approaches to the problem of providing service while partitioned and coping with inconsistencies may be found in [2].

Organization of the Echo Hierarchical Name Space

The Echo hierarchical name space is implemented in two pieces. The upper levels of the hierarchy are implemented by SRC's global name service [7, 1]. The lower levels of the hierarchy are implemented by Echo's logical file volume service. For example, consider a full path name:

```
/com/dec/research/src/users/hisgen/echo/wwos/paper.tex
```

A prefix of this name, say:

```
/com/dec/research/src/users/hisgen
```

is interpreted by the global name service. It produces an Echo mount point, which contains a (low-level) unique identifier for an Echo file service volume along with the name of the service that implements that volume. That service is accessed via remote procedure call, and the volume identifier plus the remainder of the path name are passed along to it to be interpreted.

File Volume Service Replication and Caching

Each logical file volume is implemented by a set of server CPUs and disks. The disks can be replicated and/or the server CPUs can be replicated. The components are organized in a Primary-Secondaries scheme. The Primary server CPU is chosen arbitrarily, and it then uses majority voting to determine the most up-to-date disk. In configurations with an even number of disks, witnesses are used to break ties [10].

All client reads and updates are sent to the Primary. The Primary translates file system updates into disk block writes, and sends the writes to all disks that are up and in communication. The responsibility for replication rests with the service, and not with the clients, in keeping with our assumption that clients may be Byzantine.

In general, upon any failure of a server disk or server CPU, or of the communication medium between server disks and server CPUs, a new election is held. Relatively good communication is required between the CPUs and disks that make up a single replicated service, with low latency and high bandwidth.

Our replication scheme provides a strong consistency guarantee. Inconsistencies due to partitions are ruled out by requiring a majority. By requiring all reads and updates to go through the Primary, we ensure that two clients cannot see different data, including the case of data that is in the middle of being updated. The drawback is that a majority of disks and their CPUs must be up and in communication.

Clients cache data from the file volume service. Cache consistency is achieved by having the service keep track of which clients have cached which files and directories: the Primary calls back on clients at the start of an update to a file or directory, telling them to discard that file or directory from their caches. The algorithm is similar to those of the Andrew and Sprite file systems [4, 5, 9]. The data base of which clients have which files and directories is replicated: a copy is maintained both on the Primary and on each Secondary. The file service cache consistency algorithm achieves the same strong consistency guarantee as the file service replication, that two clients cannot simultaneously see different data.

We believe that the strong consistency guarantee is important at the lower levels of the Echo hierarchical name space. First, this is where most user data will live, and the strong guarantee is less confusing to average users than a weaker guarantee would be. Second, many application programs that run in distributed computing environments were originally designed to run on single time-sharing systems where strong consistency was easily achieved and was simply taken for granted by most application programmers. Providing these programs with the semantics to which they are accustomed is easier than trying to relax the semantics and discovering by painful experience which programs no longer work.

A good example of an application program that relies on strong consistency semantics is the Unix **make** utility, which uses the existence or non-

existence of files and the last-write time-stamps on files to make its decisions. At SRC, we run our own parallel version of **make**, which takes advantage of idle machines in our local network of workstations [11].

Global Name Service Replication and Caching

SRC's global name service also employs replication for availability. Its guarantee for the consistency of its replicas is looser than that of the file volume service, but it provides greater availability.

With the global name service, updates propagate nonatomically to all replicas. The replicas eventually converge to the same value, but clients are permitted to observe intermediate states, that is, to obtain different answers to the same query from different replicas.

The global name service provides service even while the replicas are partitioned. Only one replica needs to be available for service to be provided, not a majority. Clients can obtain service from any replica; there is no notion of a primary. When a partition is reconnected, conflicting updates are resolved using timestamps: the most recent update wins.

Because the update rate is low, because the objects being updated are small (directories rather than files), and because of the way it accommodates partitions, the global name service replication scheme is better suited to geographic distribution on communication links with high latency and low bandwidth. Because of these same properties, having a larger number of replicas is feasible.

Clients cache data from the global name service, read-only. The cache consistency is weak; cache invalidation is triggered by expiration times, of say, several hours. Observe that obtaining data from an out-of-date cache is comparable to obtaining data from a name service replica that is not in communication with the other replicas.

The client cache can provide even better availability if it is always kept loaded with data that is important to this client, as determined by a combination of past history and administrative scripts, and if it times out old entries only when the client is able to communicate with a name service replica and obtain more recent data. We call these ideas *stashing:* they differ from caching in their fetch and replacement policies and in having even looser consistency.

We believe that the looser consistency guarantee of the global name service will be acceptable for the upper levels of the Echo hierarchical name space. First, much of the information at the upper levels may be structured

as hints [8]. For example, consider the mount point data, which consists of the low-level unique identifier for a volume plus the name of the service that stores it. When that service is contacted, if it no longer stores that volume, it will know it. Second, much of the data at the upper levels cannot be modified by ordinary users; its access control permits updates only by system administrators. A small number of special application programs could be written for making updates at the upper levels that would help administrators cope with inconsistency; the number of such special application programs will be much smaller than the number of general application programs. Third, the update rate at the upper levels is relatively small.

With Echo as a whole, administrators can trade off the consistency and availability of a particular directory in the hierarchical name space by deciding what service to store it in: the file volume service or the name service.

Status

Our design of Echo is almost complete, and interfaces between major components have been written. We plan to have a prototype implementation working in the summer of 1989.

References

[1] Andrew D. Birrell, Michael B. Jones, and Edward P. Wobber. A simple and efficient implementation for small databases. In *Proc. 11th Symp. on Operating Systems Principles*, pages 149–154, ACM SIGOPS, November 1987.

[2] Susan B. Davidson, Hector Garcia-Molina, and Dale Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, September 1985.

[3] Robert Hagmann. Reimplementing the Cedar file system using logging and group commit. In *Proc. 11th Symp. on Operating Systems Principles*, pages 155–162, ACM SIGOPS, November 1987.

[4] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[5] Michael L. Kazar. Synchronization and caching issues in the Andrew file system. In *Winter Conference Proceedings*, pages 27–36, USENIX Association, February 1988.

[6] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Computer Systems*, 4(3):382–401, July 1982.

[7] Butler W. Lampson. Designing a global name service. In *Proc. 5th Symp. on Principles of Distributed Computing*, pages 1–10, ACM SIGACT-SIGOPS, August 1986.

[8] Butler W. Lampson. Hints for computer system design. *IEEE Software*, 1(1):11–28, January 1984.

[9] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout. Caching in the sprite network file system. *ACM Transactions on Computer Systems*, 6(1):134–154, February 1988.

[10] Jehan-Francois Pâris. Voting with witnesses: A consistency scheme for replicated files. In *Proc. 6th International Conference on Distributed Computer Systems*, pages 606–612, IEEE Computer Society, 1986.

[11] Eric Roberts and John Ellis. Parmake and dp: experience with a distributed, parallel implementation of make. In *Proceedings from the Second Workshop on Large-Grained Parallelism*, Software Engineering Institute, Carnegie-Mellon University, Report CMU/SEI-87-SR-5, November 1987.