Granularity and Semantic Level of Replication in the Echo Distributed File System

Andy Hisgen, Andrew Birrell, Chuck Jerian, Timothy Mann,
Michael Schroeder, and Garret Swart
DEC Systems Research Center
130 Lytton Avenue
Palo Alto, CA 94301, USA
April 12, 1990

Introduction

Two major design issues in any replicated data storage system are the *semantic level of replication* and the *granularity of replication*.

By the semantic level of replication, we mean the lowest level of abstraction within the system's implementation such that the data objects stored in the replicas appear to be equal when viewed at this level. A system that presents an external interface at a particular semantic level is free in its internal implementation to replicate its data at that same semantic level or at any lower level that it chooses. For example, in a source code control system providing versioned files, the replication internally could be at the level of versioned files, but it could also be at the level of non-versioned files (with a suitable mapping from versioned file names to non-versioned), at the level of unnamed files, or even at the level of an array of logical disk blocks.

By the granularity of replication, we mean the unit of data that may be replicated independently of other units of data. For a replicated file system, examples of a unit of data include a file, a record within a file, a directory and its children, a logical volume (say, a Unix file system), or a collection of logical volumes that are co-located in the same physical resources. The design choices for granularity are constrained by the choice for the semantic level of replication: the granularity of replication may be no finer than the granularity of data objects provided by the semantic level. In general, the higher the semantic level, the more choices there are for the granularity of replication.

This paper examines the choices for semantic level and granularity that have been made in the Echo distributed file system [3, 6]. The primary goals of Echo are to explore issues of scaling, availability, and performance. For scaling and uniformity of access, Echo provides a global, hierarchical name space. 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 [2]. The log also records information about updates that are in progress, and this information is used during crash recovery to bring all replicas into agreement.

The Echo hierarchical name space is structured as a collection of subtrees, called *Echo Volumes*, that are glued together to form a single name space. Each Echo Volume may be implemented by a different service, for example, a name service or a file service [3]. In the remainder of this paper, we discuss only the file service component of Echo.

Aspects of Echo Replication

This section presents a summary of those aspects of the Echo replication algorithm that are needed for understanding the rest of the paper; other aspects are omitted.

In Echo, data storage is implemented by server computers and disks. As independent choices, an Echo hardware configuration may have replicated disks and/or multiple server computers. In general, disks are multi-ported and connected to several server computers. Each such disk has a *multi-port arbiter*, which recognizes at most one connected server computer as its *owner* at a time. Ownership is subject to timeout. Echo can also make use of single-ported disks; in this case, software on the single physically-connected server simulates the multi-port arbiter, and other logically-connected server computers access the disk via this server over the network.

Echo employs a distributed caching algorithm between clients and servers, in which servers keep track of which clients have cached what files and directories [4, 5, 7]. This caching information is replicated in the main memories of the server computers.

The server computers are organized in a primary-secondaries scheme. First, the server computers communicate amongst themselves to choose a potential primary who has up-to-date caching information.

Since server computers may be partitioned, this step can produce more than one potential primary. Next, each potential primary tries to claim ownership of a majority of the disks. If a potential primary succeeds in claiming a majority of the disks, it becomes the real primary.

Because disk ownership is subject to timeout, the current owner must refresh its ownership periodically. After a failure of the primary, a secondary must wait for the disk ownership timeouts to expire before it can become the new primary. The waiting is required in order to guarantee that there is never more than one primary. Thus, the ability to fail over quickly is dependent upon short ownership timeouts with frequent refresh.

During service, all client reads and updates are sent to the primary. For client updates, the primary writes to all disks that are up and in communication. For client reads, only one disk needs to be read, since there is at most one primary and all update traffic goes through it.

Semantic Level of Replication

In designing Echo, we considered two alternatives for the semantic level of replication, (1) at the level of files and directories, and (2) at the level of an array of logical disk blocks. Both have their advantages and disadvantages; we ultimately chose (2).

An advantage of replicating at the semantic level of files and directories is that it could facilitate reading from secondaries for load balancing (since each secondary understands how to navigate files and directories). Another advantage is that, in principle, the representation of files and directories in terms of lower-level disk blocks can be different on each replica, which could be important in an environment of heterogeneous servers or for smooth software evolution. A potential disadvantage is that, in principle, bringing the replicas into agreement during crash recovery requires that each file system operation have a corresponding undo operation, which is hard for delete operations. However, there is a special-case trick that avoids the need for general undo: all operands of the original operation may be copied from another replica that did not perform the operation.

The advantages of replicating at the semantic level of an array of logical disk blocks are as follows. It is a good match for multi-ported disk hardware: the primary writes directly to all disks without going through the secondaries, which therefore do not have to be up. The design accommodates schemes for redundant arrays of inexpensive disks (RAID) in which error-correcting codes are implemented at the block level [9]. RAID has a cost advantage over fully mirrored disks in that it provides an intermediate cost point between no redundancy and double redundancy. The log can use *new-value* logging, in which the information logged for an update is simply a sequence of records, each giving the new value for a byte range within a logical block. We believe that new-value logging is simpler to program and get right than operational logging. For example, with new-value logging, it is trivial to ensure that redo is idempotent, whereas with operational logging, redo must figure out which disk pages already have the update and and which do not. This requires careful programming and additional machinery, such as recording a log sequence number on each disk page. Another advantage is that the log is a single, common log that is replicated byte-for-byte on all disks, just like ordinary data, which makes the log less vulnerable to bad-disk-block errors. Whereas with (1) above, each replica requires its own distinct log.

Granularity of Replication

Replication schemes that provide strict consistency between replicas (i.e., one-copy serializability [1]) and that are resilient against communication partitions require the use of quorum algorithms. Schemes that choose a primary synchronization site and that maintain this primary's quorum using keep-alives (like Echo) have an ongoing expense even after entering service (The keep-alives could be done lazily, say, only if there is a client read or update request, but this still represents an expense.). Therefore, it is advantageous to employ some kind of grouping of related data, and carry out a single election for a whole group of data rather than many separate elections for individual data items. That is, the granularity of replication should be fairly large. We believe that file-level granularity is too fine, since it requires too many instances of the election algorithm.

In Echo, the granularity of replication is an entire array of logical disk blocks. A replica, called an *EchoDiskRep*, is configured from one or more physical disks. Two or more EchoDiskReps are combined to form an *EchoDisk*, which provides a replicated array of logical blocks to the next layer up. In the election, the potential primary tries to obtain ownership of a majority of the EchoDiskReps. (In configurations with an even number of EchoDiskReps, witnesses are used to break ties [8].) Echo Volumes (subtrees of files and directories) are layered on top of the EchoDisk interface. This layer is also responsible for the distributed caching algorithm between clients and servers, including replicating the caching infor-

mation in the main memories of the primary and the secondaries. Multiple Echo Volumes may be stored in a single EchoDisk, with a single instance of the election algorithm.

In general, even though a system employs a coarse granularity of replication internally, end users could still be presented the illusion of granularity at the level of files, by adding a level of indirection. At the time a file is created, it would be assigned to a particular instance of a larger-grained data unit whose properties (e.g., number of replicas) matched those desired by the end user. In Echo, we provide a weak form of this functionality. A system administrator can configure each EchoDisk to have a particular number of replicas, and can assign Echo Volumes to EchoDisks. An end user can then place a file in a volume with the desired properties (provided she has write permission to a directory in that volume). A deficiency of this scheme is that placing a file in a particular volume forces a prefix of the file name to be the name of that volume. This deficiency is mitigated by the fact that many Echo Volumes with different names can be placed in a single EchoDisk.

Status

At this writing, Echo is just beginning to turn over. The code for replication and failover has all been written, and has survived preliminary testing. Our future plans include making failover go fast.

References

[1] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[2] 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.

[3] Andy Hisgen, Andrew Birrell, Timothy Mann, Michael Schroeder, and Garret Swart. Availability and consistency tradeoffs in the Echo distributed file system. In *Proc. Second Workshop on Workstation Operating Systems*, pages 49–54. IEEE Computer Society, September 1989.

[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] Timothy Mann, Andy Hisgen, and Garret Swart. An algorithm for data replication. Technical Report 46, DEC Systems Research Center, Palo Alto, California, June 1989.

[7] 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.

[8] 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.

[9] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. of the ACM SIGMOD Conference*, pages 109–116. ACM SIGMOD Record, Volume 17, Number 3, 1988.