A Low-bandwidth Network File System

Athicha Muthitacharoen ATHICHA@LCS.MIT.EDU
Benjie Chen BENJIE@LCS.MIT.EDU
MIT Laboratory for Computer Science, 200 Technology Square, Cambridge, MA 02139 USA

David Mazières DM@CS.NYU.EDU
Department of Computer Science, New York University, 715 Broadway Room 708, New York, NY 10003 USA

1. Introduction

Wireless devices are often connected by low bandwidth and low capacity networks. While mobile users rarely consider running network file systems on wireless devices, as performance may be unacceptable, efficient remote file access would be desirable in many situations. This paper presents LBFS, a network file system designed for low bandwidth networks. LBFS exploits similarities between files or versions of the same file to save bandwidth. It avoids sending data over the network when the same data can already be found in the server's file system or the client's cache. Using this technique, LBFS achieves up to two orders of magnitude reduction in bandwidth utilization on common workloads, compared to traditional network file systems.

LBFS provides close-to-open consistency like AFS (Howard et al., 1988). Files reside safely on the server once closed, and clients see the server's latest version when they open a file. Consequently, LBFS can reasonably be used in place of other network file systems without breaking software or disturbing users.

2. Design

To save communication, LBFS uses a large, persistent file cache at the client. We designed LBFS under the assumption that clients will have enough cache to contain a user's entire working set of files (a reasonable assumption given today's storage technology). With such aggressive caching, most client/server communication is for the purpose of maintaining consistency.

LBFS reduces bandwidth requirements considerably further, by exploiting cross-file similarities. It takes advantage of the fact that the same chunks of data often appear in multiple files or multiple versions of the same file. Examples range from "auto-save" files generated by editors, to small modifications of large files made by users, to object files output by compilers after small source changes. The LBFS server indexes the file system, including recently-deleted files, to be able to find chunks of data from their hash values. The LBFS client similarly indexes a large persistent file cache. When LBFS must transfer a file between the client and server, it recognizes chunks of data the recipient already has in other files and avoids sending those chunks over the network. The target file is recreated by using chunks of the file system and client cache.

For the remainder of this section, we describe the LBFS protocol and its use of indexes.

2.1 Indexing

In order to use chunks from multiple files on the recipient, LBFS considers only non-overlapping chunks of files and avoids sensitivity to shifting file offsets by using variable-size chunks. LBFS bases chunk boundaries on file contents, rather than on position within a file. Thus, insertions and deletions only affect the surrounding chunks. Similar techniques have been used successfully in the past to segment files for the purposes of detecting unauthorized copying (Brin et al., 1995).

LBFS uses Rabin fingerprints (Rabin, 1981) to determine chunk boundaries in a file. A Rabin fingerprint of a data chunk is computed by performing a polynomial modulo of the chunk with a predetermined irreducible polynomial. LBFS computes a Rabin fingerprint on every (overlapping) 48 bytes of a file. When the low-order 13 bits of the fingerprint equal a constant value, the bytes producing that fingerprint constitute the end of a chunk, called a *breakpoint*. Rabin fingerprints are efficient to compute on a moving window in a file, making it easy for LBFS to incrementally scan and divide up files. Since the breakpoints are set according to the value of 13-bit fingerprints, assuming random data, the expected chunk size should be $2^{13} = 8192 = 8\ K$ bytes, plus the size of the breakpoint window (48 bytes). To avoid pathological cases, LBFS imposes a minimum and maximum chunk size of 2K and 64K, respectively. Such artificial suppression and creation of breakpoints can disrupt the synchronization of file

chunks between versions of a file. The risk, if this occurs, is that LBFS will perform no better than an ordinary file system. Fortunately, synchronization problems typically result from stylized files—for instance a few repeated sequences none of which has a breakpoint—and such files do well under conventional compression. Since all LBFS RPC traffic gets conventionally compressed, pathological cases do not necessarily translate into slow file access.

The LBFS client indexes chunks of all the files it has in cache. It indexes the chunks using 63-bit Rabin fingerprints. A B-tree stores mappings from fingerprint to cache file, offset, and chunk size. Similarly, the server maintains a B-tree mapping fingerprints to files in the file system, offsets, and chunk sizes. The client and server use the same set of polynomials to create chunk indexes.

Rabin fingerprints have a low but non-negligible probability of collision. Thus, we use a cryptographic hash, SHA-1, to verify the validity of data chunk when a fingerprint matches are found. We chose to index by fingerprints (rather than SHA-1 hashes) for two reasons. First, Rabin fingerprints are cheaper to compute and smaller to store. Second, we would end up recomputing SHA-1 at the time a chunk is used anyway, because LBFS allows the file system and chunk database to be out of sync. This laxness permits a file system served by LBFS to be updated locally behind the server's back. It also saves the server from performing expensive synchronous disk operations out of concern for the database's crash recovery.

2.2 Protocol

The LBFS protocol is built on top of NFS version 3. Four new RPCs are introduced: GETFP, MKTMPFILE, CONDWRITE, and COMMITTMP.

The LBFS client currently performs whole file caching. When the user opens a file, if the file is not in the local cache or the cached version is not up to date, the client issues a GETFP RPC to request a vector of $\langle \text{fingerprint}, \text{SHA-1 hash}, \text{size} \rangle$ triples. The client can look up the fingerprints in its chunk index, verify the corresponding chunks using the SHA-1 hash, and, if they match, avoid transferring the data. Several GETFP requests can be issued to cover a large file.

When transferring a dirty file to the server, the client issues a MKTMPFILE RPC. The server replies with a temporary file handle. This step is essential for two reasons. First is to guarantee that the reconstruction of the dirty file on the server side is atomic. Atomic updates limit the potential damage of simultaneous writes. Second, and more importantly, the previous version of a file will often have many chunks in common with the current version. Using a temporary file allows LBFS to utilize those chunks. To write content to the temp file handle, the client issues CONDWRITE RPCs, which are similar to NFS3 WRITE RPCs, except that, for each RPCs, the arguments contain a fingerprint and SHA-1 hash rather than the actual data to be written. If the server cannot find the chunk, CONDWRITE returns the special error code FPRINTNOTFOUND. The client will then issue a normal WRITE RPC with the actual data. Finally, the COMMITTMP RPC commits the temporary file by copying its contents to the original file.

3. Implementation

Both the client and server are implemented at user-level. The LBFS client is implemented using **xfs**, a device driver bundled with the ARLA (Westerlund & Danielsson, 1998) file system. On the server side, the LBFS server is implemented using NFS. It pretends to be an NFS client, effectively translating LBFS requests into NFS. The client and server communicate over a TCP connection, using Sun RPC. We extended an existing RPC library to compress, authenticate and encrypt the stream of RPC traffic between client and server.

4. Evaluation

We ran several experiments with LBFS. Under common operations such as editing documents and compiling software, LBFS can consume up to two orders of magnitude less bandwidth than the popular NFS file system. In measuring the end to end application performance, NFS runs much slower over 2 Mbit/sec WaveLan compared to 11 Mbit/sec mode. LBFS's performance, however, is almost identical over the low and high-bandwidth links.

References

Brin, S., Davis, J., & Garcia-Molina, H. (1995). Copy detection mechanisms for digital documents. *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data* (pp. 398–409).

Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., & West, M. J. (1988). Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, *6*, 51–81.

Rabin, M. O. (1981). *Fingerprinting by random polynomials* (Technical Report TR-15-81). Center for Research in Computing Technology, Harvard University.

Westerlund, A., & Danielsson, J. (1998). Arla—a free AFS client. *Proceedings of the 1998 USENIX, Freenix track*. New Orleans, LA.