Efficient File Storage Using Content-based Indexing

João Barreto*and Paulo Ferreira†
INESC-ID/IST
Distributed Systems Group
Rua Alves Redol Nº 9, 1000-029 Lisboa, Portugal
[joao.barreto,paulo.ferreira]@inesc-id.pt

Introduction. Content-based indexing [MCM01] is a technique of proven effectiveness for efficient transference of file contents over low bandwidth network links. Departing from this context, the natural step of extending the application of this technique to local file storage has been proposed by a number of storage solutions [CN02, QD02, BF04]. To some extent, all these solutions share a core storage model. File contents are divided into disjoint chunks of data, each of which is individually stored, along with a unique hash of its contents, in a repository of chunks. The actual files are then stored as sequences of possibly shared references to chunks in the repository.

Dividing files into smaller chunks implies a higher similarity probability, which intuitively may suggest a better storage efficiency. However, the chunk repository model entails higher storage penalties as chunk size is decreased: (a) increased chunk meta-data overhead; (b) increased internal fragmentation, if the chunk repository is stored on a block-based device; and (c) lower chunk data compression ratios are achievable. This trade-off restricts the choice of the expected chunk size to relatively high values; hence, existing solutions do not fully exploit the similarity that may exist in a file system. Furthermore, by storing chunks in a randomly organized repository, sequential read performance is penalized, even if the file being accessed does not share any chunk with other files.

We propose a novel storage architecture that eliminates the above limitations, therefore allowing for smaller chunk sizes and, accordingly, yielding higher storage gains. Moreover, it achieves this without imposing any access performance or storage penalty to files sharing no similarity with the remaining file system. We envision our solution to be especially suitable as a storage-efficient support for versioning file systems and for resource-limited embedded file systems.

Storage Architecture. Our chunk storage file system is stacked on top of a regular file system, which stores the actual file contents in secondary memory. Our architecture follows a simple principle: if a file shares no chunks with the remaining file system, it should be directly stored in the underlying file system without modification. The portion where each chunk is stored in such file is designated as its *content location*.

On the other hand, if a file does share one or more chunks with some previously stored file(s), the contents of such chunk(s) will not be stored in the underlying file system; instead, the file's metadata will include a *chunk pointer* for each such shared chunk referencing its content location (where its actual contents may be retrieved). Only the remaining unshared portion is effectively stored in the underlying file system, in the same way as described for a whole unshared file. In order to ease the maintenance of each chunk pointer when the content location of a chunk changes as result of some write access, a level of indirection is provided by a disk-stored chunk pointer table.

Hence, in case of no similarity, no storage overhead is imposed and read access performance is identical[1] to that of a regular file system. Furthermore, (a) in case of an underlying block-based file system, internal fragmentation is, on average, not affected; and (b) data compression may be applied to the unshared portions of each file as a whole, rather than to individual smaller chunks, thus achieving higher compression ratios.

Since chunk hashes are not stored along with their contents, the file system must be divided and hashed each time similar chunk detection needs to be made for new contents. This phase is deferred to background sessions, executed during idle system periods.

The organization of files into chunks employs a *chunk coalescing* step, which optimizes cases where consecutive pointers to contiguous shared chunks are detected. Such pointers are replaced by a simple multiple-chunk pointer, thus reducing the storage overhead and allowing faster access performance. In practice, this is comparable (though not always equivalent) to considering a higher chunk size whenever resorting to a lower size would yield no additional similarity gains.

Concluding Remarks. Our architecture is partially functional in a simulator and is currently being implemented as a Linux Virtual File System. Preliminary results, obtained from the analysis of actual file system contents of desktop computers at Inesc-ID show that our architecture is able to achieve higher storage reductions than any other chunk repository approach.

References

[BF04] J. Barreto and P. Ferreira. A replicated file system for resource constrained mobile devices. In *Proceedings of IADIS International Conference on Applied Computing*, 2004.

[CN02] L. Cox and B. Noble. Pastiche: Making backup cheap and easy. In *Proceedings of Fifth USENIX Symposium on Operating Systems Design and Implementation*, December 2002.

[MCM01] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *Symposium on Operating Systems Principles*, pages 174–187, 2001.

[QD02] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *First USENIX conference on File and Storage Technologies*, Monterey,CA, 2002.

*Funded by FCT Grant SFRH/BD/13859.
†Funded by FCT Project UbiRep POSI/CHS/47832/2002.

[1] With the exception of the delay induced by an additional layer in the file system stack.