BLOG: A Block Level Versioning Utility

Jake Wires (student), Mike Feeley
Department of Computer Science
University of British Columbia
{*jtwires,feeley*}*@cs.ubc.ca*

Abstract

Information is capital; disk space is a mere commodity. The paramount objective of all storage systems must therefore entail the absolute preservation of data under all circumstances, even those involving erroneous or depredatory use. Versioning file systems, such as Elephant and VersionFS, allow users to recover data from both incidental and malicious destruction, but their complexity, as well as their immanent situation within the operating system, constitute potential vulnerabilities. To provide maximum reliability, the mechanism responsible for safeguarding data durability should operate within its own trusted subsystem, impervious to problematic impingements by both users and the operating system itself.

We are developing BLOG, a block level logging and versioning utility that guarantees the durability of data written to disk, irrespective of subsequent disk operations. When used in conjunction with a virtual machine monitor like Xen, the verity of this guarantee depends only on the trustworthiness of the VMM's protective mechanisms and the stability of the underlying disk driver and hardware. Our approach is similar is some respects to other virtual machine logging systems such as ReVirt and Chronus. By operating as a pseudo-device driver in its own protected domain, BLOG ensures that the mechanism responsible for preserving data operates in hermetic isolation. All writes submitted to the disk pass through the BLOG driver en route, where they are appended to an immutable log in a file system-agnostic manner, thus facilitating interoperability with a number of popular file systems. Because BLOG delegates the complexity of managing file system semantics and version reconstruction to a user-space process, the logging mechanism, which is the crucial guarantor of data durablity, is simple enough to operate within a trusted subsystem.

BLOG logs in a copy-free manner, batching writes to disk for efficiency. File buffer caching automatically coalesces immediate overwrites, and further disk space conservation may be achieved by taking care to avoid redundant log entries. For example, even when modifying a single 512 byte sector, Linux file systems tend to write entire 4 Kbyte blocks to disk. By maintaining content summary hashes of recently read sectors (at the expense of increased memory requirements), temporal locality can redound to log entries as little as one eighth the size of unoptimized entries. The use of delayed, batched writes engenders opportunities both to discard superfluous log entries and to compress unique data before it is sent to disk, all while constraining the IO write path overhead to in-memory pointer manipulation.

Unlike Peabody and Clotho—block level systems that provide recovery at the granularity of virtual disks and volumes, respectively—BLOG will support the reconstruction of antecedent versions on a per-file basis. This file system-dependent process of reconstruction, which will not modify the critical logged data, can safely run in user space and thus will not add complexity to the trusted subsystem. However, the deliberate paucity of semantic knowledge recorded in the block level log encumbers the process of individual file reconstruction.

File reconstruction, implemented by replaying the recorded log, will rely on the well-known behavior of supported file systems. Metadata will be educed through analysis of both the content and addresses of logged sectors. For example, ext2 uses a deterministic formula for assigning inode sector addresses, allowing for inode recognition via simple address analysis. Indirect pointer blocks, however, are allocated dynamically, precluding such facile revelation. For these situations, a more involved analysis of the sectors' content will be required—in this case, entailing the verification that each potential indirect pointer is either a valid address or a special null value—before the sector can be classified as data or metadata.

To attain acceptable performance in file reconstruction, we will incorporate additional optimizations. For instance, a separate log of directory entries will be maintained to facilitate file system traversal. As well, checkpoints will be established throughout the log, again by a user-space process, to obviate the need to reconstruct files from scratch.

Although such user-space file reconstruction will require a degree of ingenuity absent in versioning file systems, it will provide an unprecedented level of security while preserving the file-level granularity abandoned by typical block level logging systems.