MAPL: Move-anywhere proximate layout

Peter Corbett, Rajesh Rajaraman, Jiri Schindler, Keith A. Smith, Pelle Wåhlström
NetApp, Inc.

1 Abstract

MAPL is a journaling file system providing advanced data management features like snapshots and clones. It is also designed to show read/write performance near that of raw disk access for both random and sequential access patterns. Further, the system is designed to show stable I/O performance over time, regardless of the workload placed on it. Important goals of the design are that MAPL seeks to make I/O to the active file system behave like I/O to a LUN on a hardware-based RAID array, while minimizing performance overhead caused by snapshots. Specifically, MAPL minimizes the effects on write activity of snapshot creation, as well as the effects of the presence of snapshots on ongoing I/O. Like the frame arrays it mimics, MAPL is optimized for large files, and its performance goals center on them.

Frame arrays that provide snapshots normally use a copyout-based scheme to enable consistent steady-state performance for sequential I/O to active (read/write) LUNs, by preserving physically sequential on-disk layout for active data. In such schemes, however, snapshot creation has a major effect on concurrent write activity, as the logical overwrite of a block, breaking the copy-on-write relation between the active and snapshot versions of data, requires the immediate copying of the old data to a new location on disk.

Snapshots in write-anywhere file systems [1, 2, 3], in contrast, have near-zero impact on performance. But write-anywhere layout can result in poor performance for sequential-read-after-random-write access patterns commonly seen in databases and certain other workloads [4].

MAPL aims to balance between these two extremes, providing efficient snapshots and near-sequential on-disk layout, even in the face of random updates. MAPL achieves these goals using three key techniques:

1. Region-based allocation preserves the physical locality of logically sequential blocks.
2. Bulk copyout of snapshot data minimizes the overhead of copying out snapshot data by batching the copyout operations.
3. The use of a per-file B-tree to organize snapshot data provides efficient indexing for the storage and retrieval of snapshot data.

1.1 MAPL file organization and snapshots

Like most file systems, MAPL represents each file as an inode. Unlike other file systems, however, MAPL inodes track both the active state of the file and the historical versions of the file belonging to various snapshots. In practice, this means that each inode maintains two different trees of blocks. A traditional tree of indirect blocks describes the live state of the file, and a B-tree contains old versions of file blocks belonging to snapshots.

1.2 Region-based allocation

For large files—MAPL's target workload—space is overprovisioned to each file in large, physically contiguous allocations called *regions*. A typical region is a few tens of megabytes. While the majority of each region holds active file data, a portion is earmarked as a *snapshot reserve*. Thus the physical size of a region is larger than the corresponding logical range of the file that maps to the region. As data blocks within the region are logically overwritten, the new data versions are written to the snapshot reserve, allowing old versions to remain available as part of the most recent snapshot.

This allocation strategy allows MAPL to preserve physically proximate layout for logically proximate data blocks. While individual data blocks may not be "in order" within a region, excellent sequential I/O performance is achieved by reading entire regions from disk and then organizing the blocks as needed in memory.

1.3 Snapshot copyout

Over time, the snapshot reserve within a region may become exhausted. At this point MAPL will perform a bulk *copyout* of the snapshot data within the region. In other words, it will read all of the blocks from the region that are not part of the active version of the file and copy them to a separate snapshot region on disk, freeing space within the original region for more new data.

By accumulating snapshot data within a region until the snapshot reserve is filled, MAPL amortizes the cost of copying the snapshot data to the snapshot region. Instead of copying a single block for each update (in the worst case), MAPL can transfer many blocks at a time, exploiting proximal I/O [4] to take advantage of the near

proximity of the copyout blocks.

The delayed copyout of snapshot data also allows data versions from short-lived snapshots to timeout within the snapshot reserve, eliminating the need to copy it.

1.4 Snapshot B-trees

In MAPL, the location of snapshot data changes over time as it is copied from active file system regions into dedicated snapshot regions. To track these changes, MAPL keeps a B-tree of copied out data for each inode. MAPL sorts entries in these B-trees first by file block number and then by snapshot ID.

Snapshot IDs are integers that strictly increase over time. This allows the snapshot ID in a B-tree key to be considered as a ”valid until” ID. Snapshot blocks are thus looked up through inexact matching of B-tree keys. If a block is sought in the snapshot with ID n, a single lookup can determine whether the block exists either in that snapshot or in an earlier snapshot (i.e., the version with the closest snapshot ID less than n).

2 Poster Presentation

Keith Smith will present this poster.

References

[1] Valerie Aurora. A short history of btrfs. `http://lwn.net/Articles/342892`, Jul 2009.

[2] Dave Hitz, James Lau, and Michael Malcolm. File system design for an NFS file server appliance. In *Proceedings of USENIX Winter 1994 Technical Conference*, pages 235–246, Jan 1994.

[3] Sun Microsystems. ZFS at OpenSolaris community. `http://opensolaris.org/os/community/zfs/`.

[4] Jiri Schindler, Sandip Shete, and Keith A. Smith. Improving throughput for small disk requests with proximal I/O. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies*, Feb 2011.