The Expandable Network Disk

Athicha Muthitacharoen Robert Morris M. Frans Kaashoek
{athicha,rtm,kaashoek}@csail.mit.edu

Many organizations posses large data sets that need to be online but do not require the highest possible performance. It often makes sense to store this data on low-end commodity servers or on idle disk space of servers primarily used for something else (for example, compute servers). Once the amount of storage required grows beyond a few disks' worth, a cluster storage system starts to be attractive. An ideal cluster storage system would present a disk-like interface compatible with existing file systems (perhaps via iSCSI), keep running despite the failure of a few bricks, recover quickly after correction of a temporary failure that does not disturb disk contents, and expand transparently to file systems.

Efficient handling of temporary failures is particularly important when the cluster is composed of non-dedicated bricks that may be rebooted due to the needs of their primary function, or when large collections of inexpensive bricks have frequent failures of parts such as power supplies and network connectivity. A temporary brick failure poses two questions. First, how will redundancy be maintained for data written during the failure that would ordinarily be replicated on the failed brick? Second, when the failed brick recovers, how will the system know which of the blocks it stores are out of date, and which can safely be used? Petal [1] and FAB [2], two existing cluster storage systems, do not maintain full redundancy during temporary failures, and restore redundancy by copying newly written blocks to a recovering brick. A primary contribution of this paper is to maintain full redundancy during temporary failures and to avoid the expense of having to bring a recovered brick up to date.

This abstract presents END (the Expandable Network Disk). END appears to a file system as a giant physical disk, accessible through a standard block-level I/O interface; as a result, END supports existing disk file systems without modification. END only commits physical storage as blocks are written, so that its apparent address space can be much larger than the amount of physical storage. Each file system stored in END would also be created with a size large enough to accommodate future expansion. Thus, as storage needs increase over time, an administrator can add new bricks and existing file systems will automatically have access to more storage.

END uses a two-layer design, in which storage "bricks" hold two kinds of information. The lower layer stores replicated immutable "chunks" of data, each indexed by a key equal to a hash of its content. The upper layer maps each block address to the key of its current content; each mapping is held on two bricks using primary-backup replication. A brick that stores the address mapping of a given block typically does not also store the chunk holding the block's current content. This separation gives END the flexibility to store chunks on whatever brick is convenient; the immutability of chunks means that END can retrieve the chunk for a key from any brick without worrying about whether it has been kept up to date.

Several benefits spring from END's separation of the mutable address mappings from the content-addressed storage of immutable data. First, the separation allows fast recovery from failures. END needs to move only the address mappings among bricks during reconfiguration, not the data; and since the mappings are small compared to the data, reconfiguration is quick. Second, END can handle temporary brick failures efficiently. If a brick fails but soon recovers with disk contents intact, END can immediately take advantage of the data chunks stored on the brick's disk. Since the chunks are immutable, they cannot get out of date even if the corresponding blocks were written while the brick was off-line. Third, END can store content-addressed data chunks on any brick, so it can always fully replicate written data, even while bricks are off-line. Fourth, this flexibility allows END to spread the load away from bricks with full disks. Finally, a newly added brick can gradually pick up its share of data chunks. It does not need to copy all the chunks in an address range to start serving requests.

Experiments with our END prototype show that bulk write operations on a file system stored on an END system of 4 bricks has slightly better performance than on a single-disk file system. As expected, END's performance is worse for small file system operations. END's bulk write throughput scales linearly with the number of bricks, and reaches 31MB/s with 10 bricks. END recovers from a single brick failure in less than two minutes; writes completed afterwards are fully replicated. END reincorporates a rebooting brick in approximately one minute and is able to use the chunks already stored on the brick prior to its reboot. During a bulk write workload, adding a new brick to END completes in about one minute.

References

[1] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *ASPLOS*, pages 84–92, Cambridge, MA, 1996.

[2] Y. Saito, S. Frolund, A. Veitch, A. Merchant, and S. Spence. FAB: building distributed enterprise disk arrays from commodity components. In *ASPLOS*, pages 48–58, 2004.