NexentaStor: ZFS Initialization and Resilvering

A common whiteboard session captured on paper

Chris Nelson, Sales Engineer, Nexenta Systems

May 2012

NexentaStor: ZFS Initialization and Resilvering

Introduction

This paper started as an explanation of the term "copy on write" as it applies to ZFS, but the foundational concepts relevant to that discussion are equally applicable to many other great features of ZFS as well, so this paper was expanded and then broken apart to cover a broader selection of ZFS highlights often described in a whiteboard session. This third paper builds upon the first two by describing how ZFS handles initialization of RAID sets and resilvering after disk failures differently from most traditional storage vendors.

Read First: ***"NexentaStor: ZFS Snapshots and Replication"***

ZFS Resilvering Offers Integrity and Efficiency

Much like replication, resilvering with ZFS technology is efficient due to its file system layout and its use of birth times. More specifically, there are three traditions used by most file systems upon which ZFS improves significantly:

1. **Creating a New RAID Set:** In traditional file systems, when a new mirror (e.g., RAID-1 or RAID-10) or a new RAID stripe (e.g., RAID-5 or RAID-6) is created, the disks have to be initialized—either with all zeroes or by copying one disk to the other (or XOR-ing them together). The goal of this is that all of the disks in the RAID set are self-consistent, but the reality is that the data on the drives is irrelevant! Instead of this wasted effort, as depicted in Figure 1, **ZFS only copies "live," or relevant, blocks of data** when creating mirrors or RAID groups. This means that **it takes nearly zero time to create and "initialize" new RAID sets!** Compare that to the time required to initialize disks reaching 3TB today and 10TB in the near future.

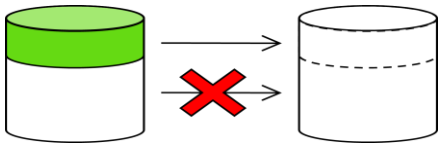

Figure 1. ZFS only copies useful data—not unnecessary garbage.

2. **Recovering from a Transient Outage:** Many traditional file systems use a concept called "dirty region logging" to keep track of changes that occur during a transient outage (e.g., from a temporarily pulled cable or interruption on a SAN). If all of the changes are physically close together on disk, this can work, but it can be slow and does not work well at all when changes are of a more random nature. As one of the early ZFS engineers put it, "outages happen for periods of time; they don't occur for periods of space." Instead of dirty region logging, **ZFS uses a more efficient concept aptly named "dirty time logging".** For each drive in a storage pool, ZFS keeps track of which transaction groups have been applied, so—if a drive is offline for a period of time—the same birth time comparison used in replication (see Figure 2) is used to identify what parts of the file system changes need to be applied to the drive when it comes back online. The net result is that **a 10-second outage takes only about 10 seconds to repair!**

Copyright © 2012 Nexenta® Systems, ALL RIGHTS RESERVED

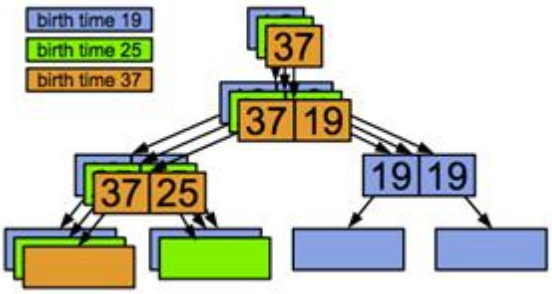

Figure 2. Using birth times makes it simple to determine what has changed between two points in time.

3. **Replacing a Failed Drive:** Much like the creation of a new RAID set, when a drive is replaced in traditional file systems, the entire drive is reconstructed from the surviving members of the RAID set (either a mirrored disk or multiple disks in something like a RAID-5 set)—even if the failed disk had only a few bytes of relevant data. Imagine the time required and performance impact of reconstructing 3TB of data when only 50MB mattered! Beyond that, traditional file systems reconstruct drives from one end to the other without any thought to the importance of the data on the disk (consider if the root directory inode was one of the last things to be reconstructed on a replacement drive, and a second disk failure occurred before the reconstruction completed…big trouble!).

   And perhaps worse than the wasted effort and non-prioritized order, traditional file systems have no significant means to check the validity of the reconstructed data along the way. (Recall the discussion of the RAID-5 write hole: If a data update had completed without the associated parity update, the reconstruction process would use the incorrect parity to "correct," i.e., corrupt, the data on the replacement drive.)

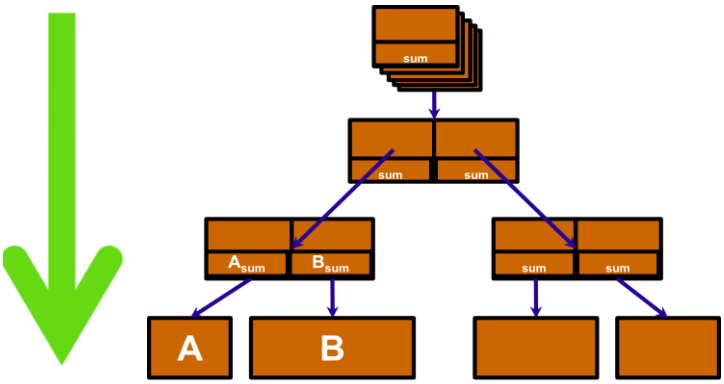

Figure 3. ZFS resilvers the most important parts of the file system first, "top-down".

Copyright © 2012 Nexenta® Systems, ALL RIGHTS RESERVED

ZFS addresses these three issues by resilvering "top-down" from the most important blocks in its tree to the least—only reconstructing blocks that matter and writing those on the new drive, and it verifies the validity of every block read using its checksums, stored in the "parent" blocks along the way. This is extremely significant for both efficiency and data integrity—especially as drives continue to grow.

Consider the process of rebuilding a drive today and how many terabytes of data must be read from other drives to do so. Worse yet, consider the ever larger drives expected in the coming years. Then, consider that enterprise-class drive specifications claim uncorrectable bit-error rates of around 1 in10^{15} bits (about 1 bit in every 120TB read) but that the more realistic error rates reported from drive manufacturers are to be around about 1 bit in every 8-20TB read. If reconstructing a disk requires more than 8TB of data be read, one can *expect at least one instance of garbage data* will be used in the reconstruction. This only highlights the need for data integrity at the level only ZFS can offer.

Read Next: *Stay tuned to nexenta.com for more!*

Copyright © 2012 Nexenta® Systems, ALL RIGHTS RESERVED