NexentaStor: ZFS Snapshots and Replication

A Common Whiteboard Session Captured on Paper

Chris Nelson, Sales Engineer, Nexenta Systems

May 2012

NexentaStor: ZFS Snapshots and Replication

A common whiteboard session captured on paper

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 second paper builds upon the first by describing how ZFS handles snapshots and replication differently from most traditional storage vendors.

Read First: *“NexentaStor: ZFS Copy-on-Write, Checksums, and Consistency”*

ZFS Snapshots are Simple!

Because of what was described in the previous paper, the concept of snapshots becomes exceedingly simple. In the following diagram, if a snapshot of the all-orange state of the file system were desired, ZFS would just skip the step of freeing all of the old orange blocks and the orange uberblock. (In concept, taking a snapshot actually is less work for ZFS, because it skips the step of freeing old blocks!) Finding data in the current file system is as easy as following the relevant pointers from the green uberblock, and finding data in the snapshot is simply following the relevant pointers from the orange uberblock. Thus, there is no need for separate “snapshot space” in ZFS, and snapshots only consume as much space as is changed in the file system since the time the snapshot was taken.

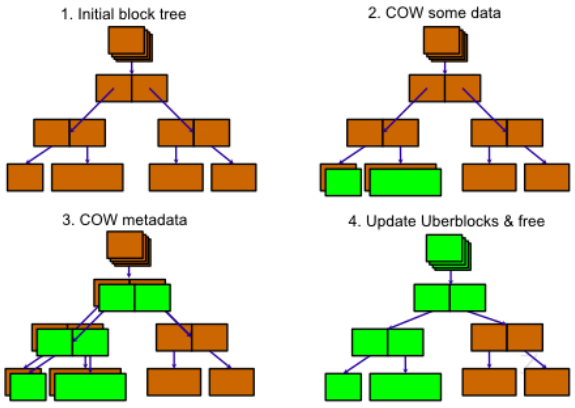

Figure 1. ZFS updates are atomic—the file system state always is consistent.

It is worth observing here how ZFS's application of "copy on write" in regards to snapshots is different than many other storage snapshot technologies in the industry. For some, when writes occur after a snapshot, the original data is literally copied into a special snapshot space before the new write is completed in the original location. For others, new writes after a snapshot are directed into the special snapshot space, and when the snapshot is deleted, those writes are merged back into the original locations. An IBM article (http://www.ibm.com/developerworks/tivoli/library/t-snaptsm1/index.html) describes several of the traditional storage vendors' snapshot strategies; note that **none of these are as efficient as ZFS.**

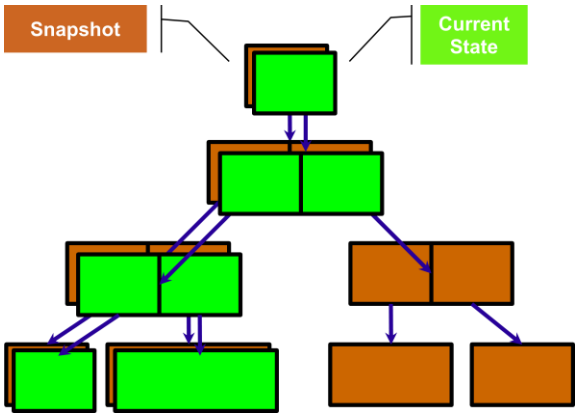

Figure 2. For ZFS, snapshots are simply keeping some pointers that would have otherwise been discarded.

ZFS Replication is Simple and Efficient!

The simplicity of ZFS snapshots—as a result of the way updates are applied to a ZFS data structure—makes understanding ZFS's built-in replication feature (known as "ZFS send and receive") simple, as well. An understanding of only one additional concept is needed—that of birth times. In addition to checksum and pointer information, each "parent" block contains the "birth times" of the children to which it points. (The birth time simply is the transaction group number during for the block that was written.) With birth times, it becomes a trivial task to identify which changes were made between times 19 and 37, for example, as in Figure 3.

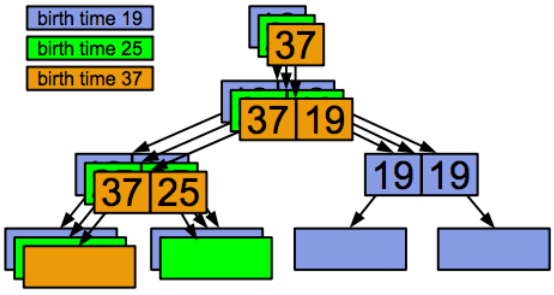

Figure 3. Using birth times makes it simple to determine what has changed between two points in time.

In this figure, if ZFS is to replicate the updates between snapshots taken at time 19 and time 37, the logical process would be as follows: Starting from the orange uberblock, the question asked is whether the child's birth time (37) is greater than 19? Yes, it is. So the state of the file system has changed, and those changes should be found and sent. Moving down and left first, the question asked is whether the child's birth time (37 again) is greater than 19? Yes. Again, down and left; is 37 greater than 19? Yes, so send that bottom-left orange data block, then back up one level and move right; is 25 greater than 19? Yes, so send that green data block, then back up one level and move right; is 19 greater than 19? No, so nothing on the right side has changed, and there is no need to traverse any further into that part of the file system. (Note that any parent block always is as "new" as its newest child.)

It also is worth noting that only the data in those data blocks are sent—not the various block pointers. Those data changes are applied on the target file system, but the block pointer tree *may be different!* **This allows for differing compression levels on source and target, or deduplication choices, or any of the other ZFS metadata-related configurable options!** (For NexentaStor, this is effective as of version 3.1.3.)

This differs significantly from other replication technologies that require complete bitmaps of a file system and comparisons of the entire file system state in order to know what changes to send. Rsync, for example, must examine the entire file structure on both source and target to determine if anything has changed; this can result in hours of metadata analysis even to send changes to one tiny file! So, with birth times and the ZFS's standard methods of applying changes to the file system, replication is both simple and efficient.

Note: NexentaStor's Auto-Sync replication feature uses ZFS's built-in send and receive technology, as described in this paper. Auto-Tier uses rsync, which, while less efficient, offers the ability to replicate between NexentaStor and non-ZFS storage.

Read Next: *"NexentaStor: ZFS Initialization and Resilvering"*