An Introduction to the Implementation of ZFS

Brought to you by

Dr. Marshall Kirk McKusick

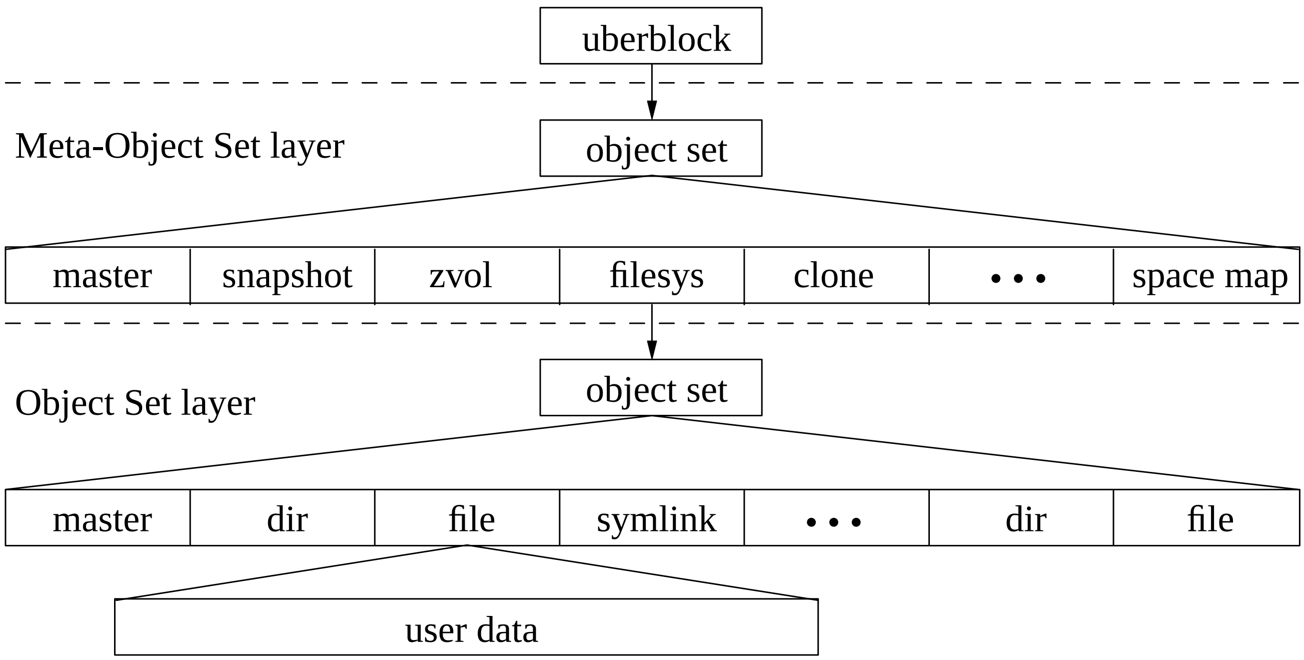
European BSD Conference 2014
27th September 2014

InterExpo Congress Center
Sofia, Bulgaria

Zettabyte Filesystem Overview

- Never over-write an existing block
- Filesystem is always consistent
- State atomically advances at checkpoints
- Snapshots (read-only) and clones (read-write) are cheap and plentiful
- Metadata redundancy and data checksums
- Selective data compression and deduplication
- Pooled storage shared among filesystems
- Mirroring and single, double, and triple parity RAIDZ
- Space management with quotas and reservations
- Fast remote replication and backups

Structural Organization



- Uberblock anchors the pool
- Meta-object-set (MOS) describes array of filesystems, clones, snapshots, and ZVOLs
- Each MOS object references an object-set that describes its objects
- Filesystem object sets describe an array of files, directories, etc.
- Each filesystem object describes an array of bytes

ZFS Block Pointer

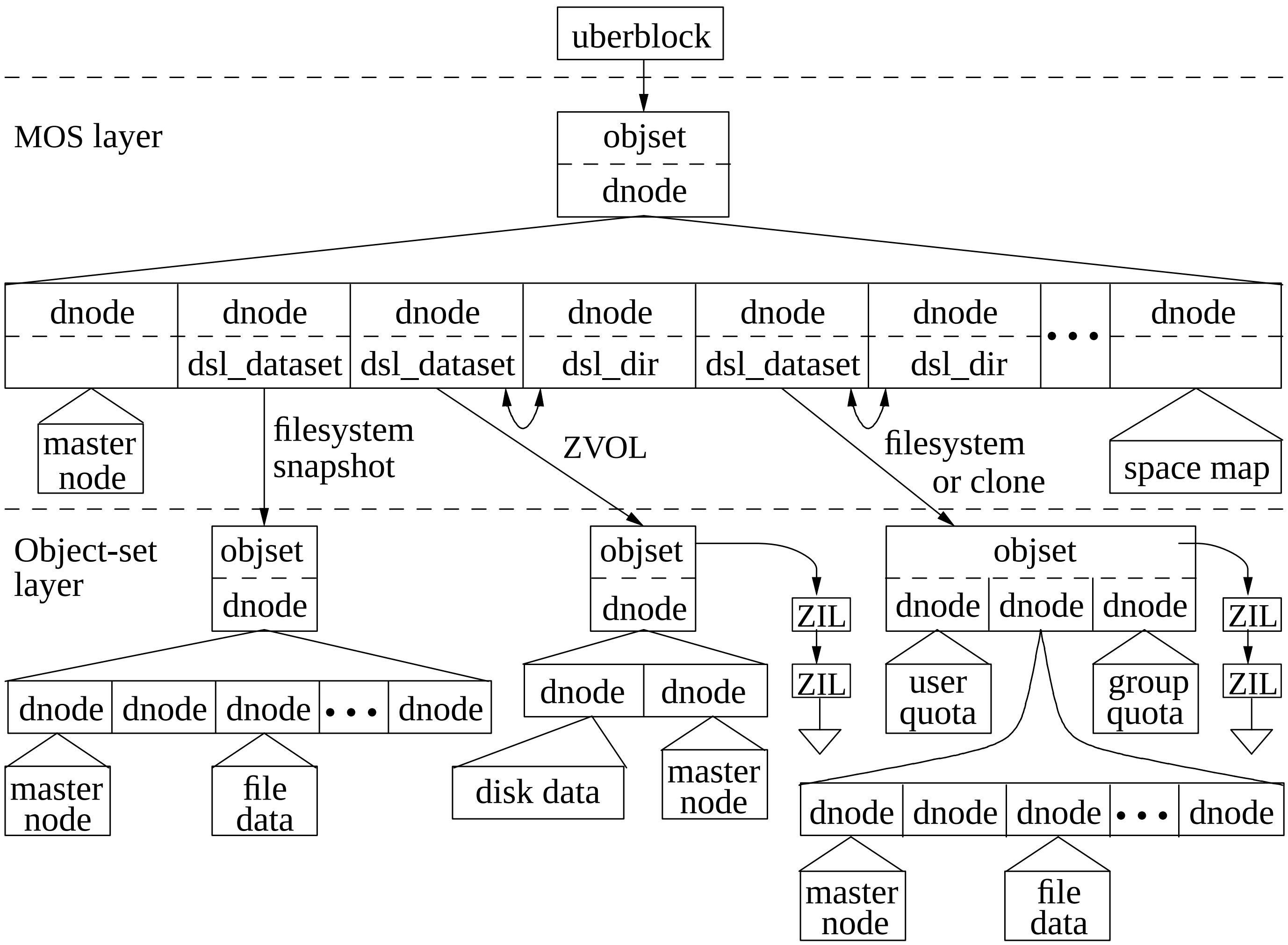
<table>
<tr><th></th><th colspan="4">64</th><th>56</th><th>48</th><th>40</th><th>32</th><th>24</th><th>16</th><th>8</th><th>0</th></tr>
<tr><td>0</td><td colspan="7">vdev1</td><td>grid</td><td colspan="3">asize</td></tr>
<tr><td>1</td><td>G</td><td colspan="10">offset1</td></tr>
<tr><td>2</td><td colspan="7">vdev2</td><td>grid</td><td colspan="3">asize</td></tr>
<tr><td>3</td><td>G</td><td colspan="10">offset2</td></tr>
<tr><td>4</td><td colspan="7">vdev3</td><td>grid</td><td colspan="3">asize</td></tr>
<tr><td>5</td><td>G</td><td colspan="10">offset3</td></tr>
<tr><td>6</td><td>B</td><td>D</td><td>X</td><td>lvl</td><td>type</td><td>cksum</td><td>comp</td><td colspan="2">psize</td><td colspan="2">lsize</td></tr>
<tr><td>7</td><td colspan="11">spare</td></tr>
<tr><td>8</td><td colspan="11">spare</td></tr>
<tr><td>9</td><td colspan="11">physical birth time</td></tr>
<tr><td>A</td><td colspan="11">logical birth time</td></tr>
<tr><td>B</td><td colspan="11">fill count</td></tr>
<tr><td>C</td><td colspan="11">checksum[0]</td></tr>
<tr><td>D</td><td colspan="11">checksum[1]</td></tr>
<tr><td>E</td><td colspan="11">checksum[2]</td></tr>
<tr><td>F</td><td colspan="11">checksum[3]</td></tr>
</table>

- Up to three levels of redundancy
- Checksum separate from data
- Birth time is the transaction-group number in which it was allocated
- Maintains allocated, physical (compressed), and logical sizes

ZFS Block Management

- Disk blocks are kept in a pool
- Multiple filesystems and their snapshots are held in the pool
- Blocks from the pool are given to filesystems on demand and reclaimed to the pool when freed
- Space may be reserved to ensure future availability
- Quotas may be imposed to limit the space that may be used

ZFS Structure



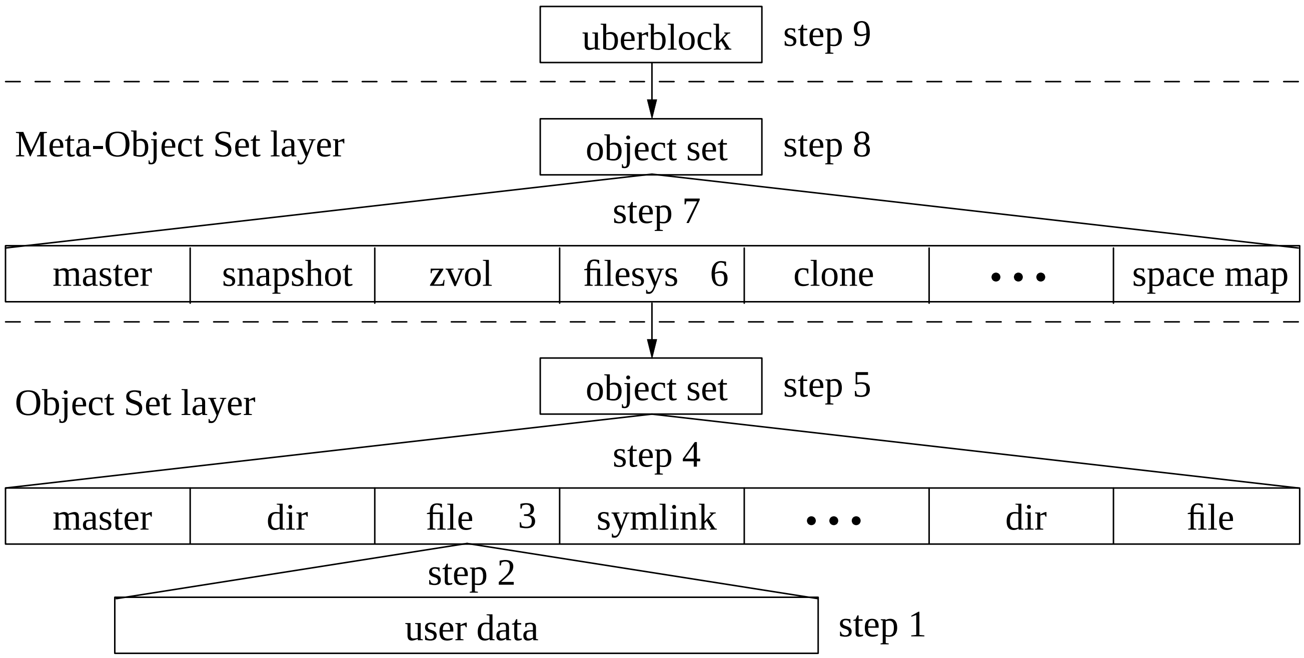
- MOS layer manages space and objects using that space

- Object-set layer manages filesystems, snapshots, clones, and ZVOLs

ZFS Checkpoint

- Collect all updates in memory
- Periodically write all changes to an unused location to create a checkpoint
- Last step in checkpoint writes a new uberblock
- Entire pool is always consistent
- Checkpoint affects all filesystems, clones, snapshots, and ZVOL in the pool
- Need to log any changes between checkpoints that need to be persistent
- The **fsync** system call is implemented by forcing a log write not by doing a checkpoint
- Recovery starts from last checkpoint, rolls forward through log, then creates new checkpoint

Flushing Dirty Data



Write modified data in this order:

1) new or modified user data

2) indirect blocks to new user data

3) new dnode block

4) indirect blocks to modified dnodes

5) object-set dnode for filesystem dnodes

6) filesystem dnode to reference objset dnode

7) indirect blocks to modified meta-objects

8) MOS object-set for meta-object dnode

9) new uberblock (plus its copies)

ZFS Strengths

- High write throughput
- Fast RAIDZ reconstruction on pools with less than 40% utilization
- Avoids RAID "write hole"
- Blocks move between filesystems as needed
- Integration eases administration (mount points, exports, etc)

ZFS Weaknesses

- Slowly written files scattered on disk
- Slow RAIDZ reconstruction on pools with greater than 40% utilization
- Block cache must fit in the kernel’s address space, thus works well only on 64-bit systems
- Needs under 75% utilization for good write performance
- RAIDZ has high overhead for small block sizes such as 4 Kbyte blocks typically used by databases and ZVOLs.
- Blocks cached in memory are not part of the unified memory cache so inefficient for files and executables using **mmap** or when using **sendfile**

Questions

More on ZFS:

- ‘‘The Design and Implementation of the FreeBSD Operating System, 2nd Edition’’, Chapter 10
- Manual pages: zfs(8), zpool(8), zdb(8)

Marshall Kirk McKusick

<mckusick@mckusick.com>

http://www.mckusick.com