0.6.3

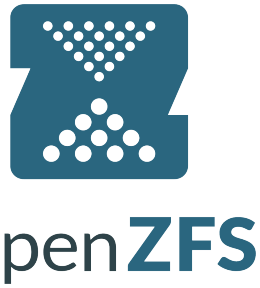
use your disks in best possible ways

Dobrica Pavlinušić
http://blog.rot13.org
DORS/CLUC 2014-06-17

What are we going to talk about?

- ZFS history
- Disks or SSD and for what?
- Installation
- Create pool, filesystem and/or block device
- ARC, L2ARC, ZIL
- snapshots, send/receive
- scrub, disk reliability (smart)
- tuning zfs
- downsides

ZFS history



2001 – Development of ZFS started with two engineers at Sun Microsystems.

2005 – Source code was released as part of OpenSolaris.

2006 – Development of FUSE port for Linux started.

2007 – Apple started porting ZFS to Mac OS X.

2008 – A port to FreeBSD was released as part of FreeBSD 7.0.

2008 – Development of a native Linux port started.

2009 – Apple's ZFS project closed. The MacZFS project continued to develop the code.

2010 – OpenSolaris was discontinued, the last release was forked. Further development of ZFS on Solaris was no longer open source.

2010 – illumos was founded as the truly open source successor to OpenSolaris. Development of ZFS continued in the open. Ports of ZFS to other platforms continued porting upstream changes from illumos.

2012 – Feature flags were introduced to replace legacy on-disk version numbers, enabling easier distributed evolution of the ZFS on-disk format to support new features.

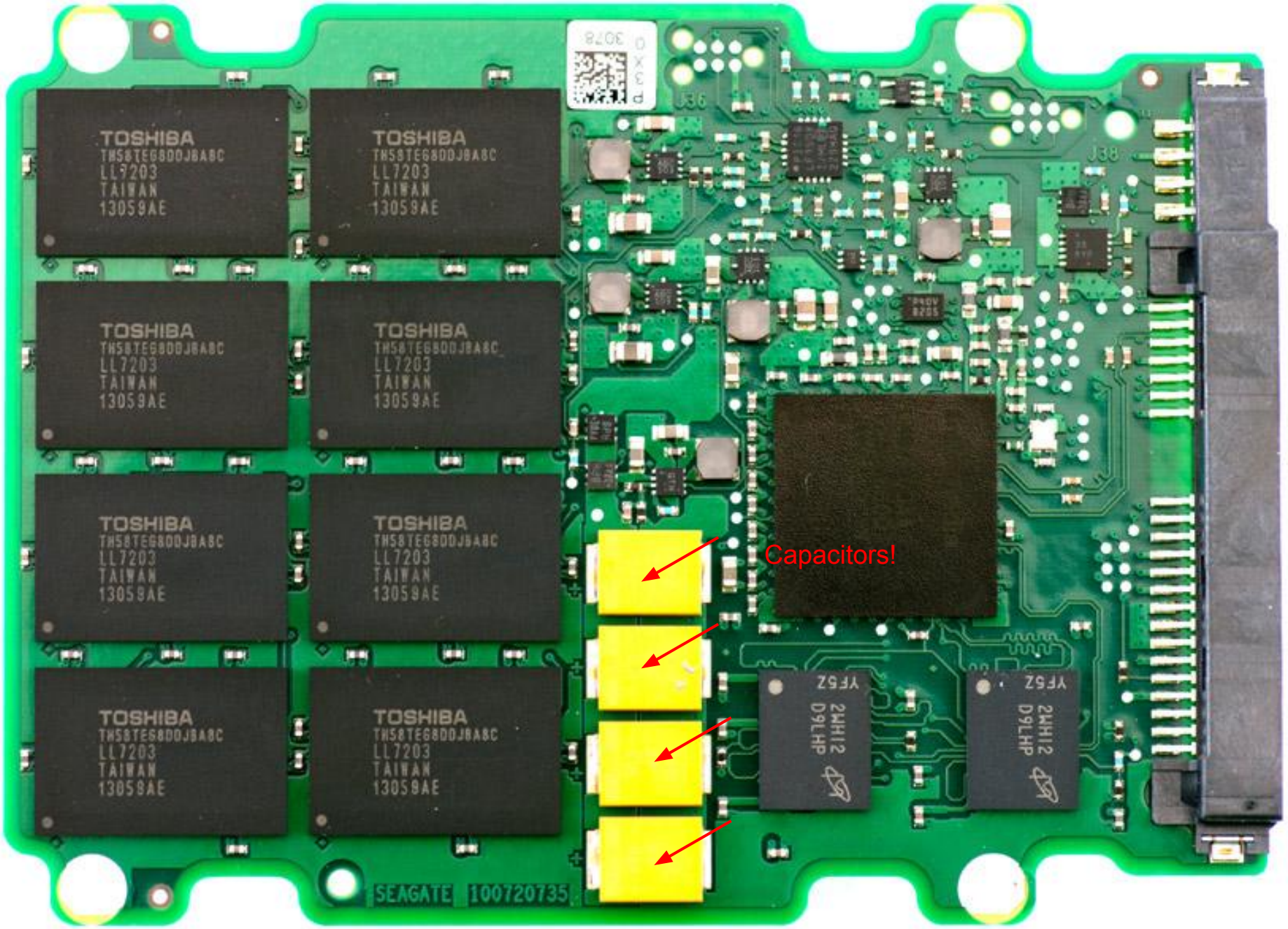
2013 – Alongside the stable version of MacZFS, ZFS-OSX used ZFS on Linux as a basis for the next generation of MacZFS.

2013 – The first stable release of ZFS on Linux.

2013 – Official announcement of the OpenZFS project.

Terminology

- COW - copy on write
  - doesn’t modify data in-place on disk
- checksums - protect data integrity
- vdev - disks with redundancy
- pool - collection of vdevs with fs or block dev
- slog - sync log to improve COW performance
- arc - RAM based cache (ECC memory recommended)
- l2arc - disk/SSD cache for arc spill over


TOSHIBA
TH58TEG8DDJBA8C
LL7203
TAIWAN
13059AE
Capacitors!
SEAGATE 100720735

Disks or SSD?

- ZFS is designed to use rotating platters to store data and RAM/SSD for speedup
- Use JBOD disks and non-RAID controllers!
- Use disks which have fast error reporting (older WD green with firmware upgrade)
- SSD with capacitors **required** for SLOG or VDEVs if you care about your data!
- ZoL doesn’t use TRIM (sigh!) - overprovision SSD for durability (80% capacity for vdev, 10% for SLOG)

Which kernel?

- x86_64 (for i386 use zfs-fuse ;-), 32-bit ARM support under development (for NAS boxes)
- 3.2.0 (wheezy) or later
- Voluntary Kernel Preemption

```
arh-hw:~# grep CONFIG_PREEMPT_VOLUNTARY /boot/config-3.2.0-4-amd64
CONFIG_PREEMPT_VOLUNTARY=y
```

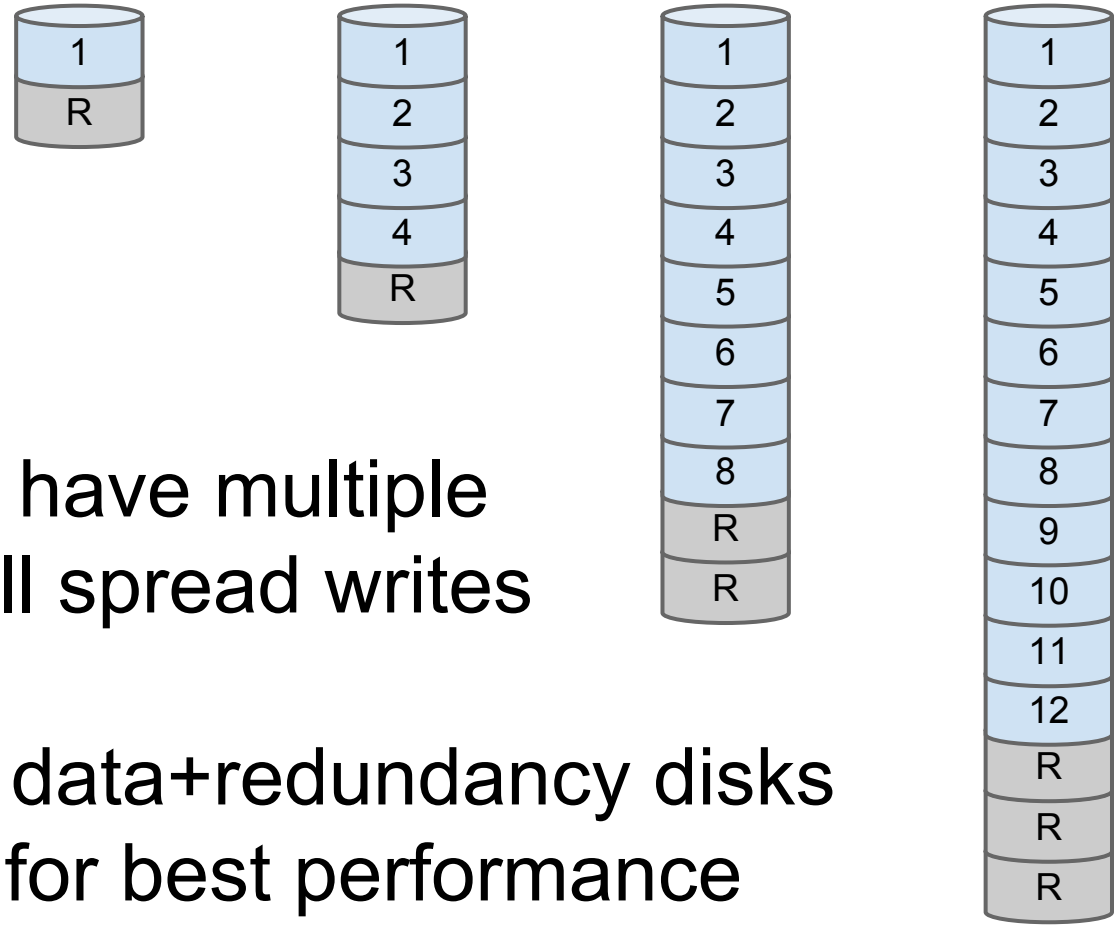
- http://zfsonlinux.org/debian.html
  - DKMS, in Debian experimental
- ZoL git production ready, needed for 3.12+
- ZFS CDDL incompatible with GPLv2 - it will be out-of-tree forever!

Booting from ZFS

- **Does not work from RAIDZ**
  - needs plain disk or mirror for boot to work!
- http://wiki.complete.org/ZFSRescueDisc
- http://people.debian.org/~jgoerzen/rescue-zfs/
- https://github.com/zfsonlinux/pkg-zfs/wiki/HOWTO-install-Debian-GNU-Linux-to-a-Native-ZFS-Root-Filesystem
- https://github.com/zfsonlinux/pkg-zfs/wiki/HOWTO-use-a-zvol-as-a-swap-device
- TL;DR - install root and boot somewhere else

ZFS redundancy options

- vdev
  - mirror
  - RAIDZ1
  - RAIDZ2
  - RAIDZ3
- each pool can have multiple vdevs, ZFS will spread writes over them
- mirrors or 2^n data+redundancy disks in single vdev for best performance



ZFS RAIDZ stripe width

How I Learned to Stop Worrying and Love RAIDZ

- http://blog.delphix.com/matt/2014/06/06/zfs-stripe-width/
- TL;DR: Choose a RAID-Z stripe width based on your IOPS needs and the amount of space you are willing to devote to parity information.
- random IOPS - use small number of disks in each RAID-Z group (group has 1 disk performance!)
- reliability - more parity (RAIDZ3 instead of RAIDZ1), groups to match storage hardware
- space efficiency - use a large number of disks in each RAID-Z group
- space efficiency - doubling the number of “data” disks will halve the amount of parity per MB of data
- Use RAID-Z. Not too wide. Enable compression.

Create pool

`zpool create -o ashift=12 tank ([raidz1-3] /dev/disk/by-id/…) ...`

- ashift=12 (align to 4k boundary)
- You **WILL NOT be able to shrink pool!**
- **use /dev/disk/by-id/ to create pool!**

`zpool history [-il]` see what your pool did!

use sparse files to create degraded pool

```
dd if=/dev/zero of=/zfs1 bs=1 count=1 seek=512G
zfs offline <pool> /zfs1
```

Create file system or volume

Turn compression on!

```
zfs set compression=lzo4 <pool|fs>
zfs create <pool>/fs
zfs create -V 512M <pool>/block/disk1
zfs set primarycache=none <pool>
```

- Don’t use dedup, even devs don’t like it :-)
  - ~500 bytes of memory per block, CPU overhead due to hashing, non-linear access to data

Volumes

- You can have zfs pool inside zfs volume!
- /dev/zvol/pool/vol - volumes and partitions (after partprobe or kpartx)
- thin-provision images for kvm

```
zfs create -V 100500G -s -b 128K -o compression=lz4 archive/zvol
```

ARC, L2ARC - read cache

ARC uses ~50% of RAM available!

```
cat /proc/spl/kstat/zfs/arcstats or arcstat.pl
```

L2ARC - Any SSD is good enough for it, you might use /dev/zram for it to get compression!

khugepaged eats 100% CPU?

```
echo 0 > /sys/kernel/mm/transparent_hugepage/khugepaged/defrag
echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

Possible to use zram for L2ARC to get compression!

L2ARC headers must fit in ARC (RAM)!

ZIL - (sync)log - NOT write cache!

- put log on separate device!
- ZFS assumes it's fastest storage (battery backed RAM, SLC SDD, mirror it!)
- `logbias = throughput`
- `sync = always` if you have slog device!
- zil_slog_limit - log/vdev target split
  - 1 Mb => idea is to keep slog always fast
- does NOT play nice with iSCSI write-back

snapshots

- Copy on write semantics (LVM isn't!)
  - Point in time view of filesystem
- Can be cloned to create writable copy
  - and promoted to master copy -> rollback!

zfs create filesystem@snapshot

zfs rollback filesystem@snapshot2

zfs list -t snapshots filesystem

zfs set snapdir=visible filesystem

zfs clone snapshot filesystem|volume

zfs diff snapshot snapshot|filesystem

zfs send/receive

- Snapshot filesystem or full pool
- Transfer (incremental) snapshot to another pool -> disaster recovery
  - this will uncompress your data, have enough CPU!
- slow transfer (~10 Mb/s) on fragmented pools with lot of snapshots
- LVM snapshots, rsync and shell script from hell or snapshot manager
  - http://sysadmin-cookbook.rot13.org/#zfs
  - https://github.com/bassu/bzman
  - https://github.com/briner/dolly
  - https://code.google.com/p/zxfer/

Disk reliability

- Disks **will** fail! That's why we are using ZFS (or some RAID) in the first place!
- zpool scrub pool
  - at least weekly, that's what checksums are for!
  - if disk disappear during scrub, and comes back after reboot it will automatically resilver data
  - scrub create random IO - it will impact performance

http://en.wikipedia.org/wiki/ZFS#Error_rates_in_hard_disks

http://en.wikipedia.org/wiki/ZFS#Silent_data_corruption

- tell kernel that device died:

```
echo 1 > /sys/block/<sdX>/device/delete
```

(not so)smart - better than nothing

- Lies, damn lies and smart counters! http://research.google.com/pubs/pub32774.html
- smartctl -t long /dev/sd? weekly or monthly
- Log output of all drives (and controllers!) and store it in git for easy git log -p http://sysadmin-cookbook.rot13.org/#dump_smart_sh
- Look out for write counters on SSD to detect wearout
- Check error recovery with smartctl -l scterc /dev/sdx
  - newer disks disable that in firmware (sigh!)
  - http://idle3-tools.sourceforge.net/
- Relocate known bad sectors
  - http://sysadmin-cookbook.rot13.org/#smart_test_relocate_pl
- Intel SSD report NAND_Writes_1GiB

IOPS - ZFS tuning - zpool iostat -v

- mirrors
  - always faster than any RAID (1-disk perf!)
  - read-only load which doesn't fit in ARC
- RDBMS - tune recordsize, logbias, primarycache=metadata

You shouldn't need to tune zfs, but...

```
cat /etc/modprobe.d/zfs.conf
options zfs zfs_nocacheflush=1 zfs_arc_max=154618822656 zfs_arc_min=1073741824
```

meta-data heavy workloads (rsync)

- increase `/sys/module/zfs/parameters/zfs_arc_meta_{limit,prune}`
- `zfs set primarycache=metadata <fileystem>`

http://www.nanowolk.nl/ext/2013_02_zfs_sequential_read_write_performance/

ZFS downsides

- out-of-kernel due to CDDL
  - DKMS and distribution supports mitigate this
- performance not main goal
  - xfs is still fastest Linux fs, run it on RAID!
- not ready for SSD pools without TRIM
- doesn’t support shrinking of pool
- you can’t remove dedup metadata
- doesn’t have rebalance (as btrfs does)
  - zfs send/receive as workaround
- storage appliance model due to memory usage vs mixed workload servers
  - doesn’t support O_DIRECT -> double buffering

References

- OpenZFS web site http://open-zfs.org/
- zfs-discuss mailing list http://zfsonlinux.org/lists.html
- ZFS on Linux / OpenZFS presentation http://events.linuxfoundation.org/sites/events/files/slides/OpenZFS%20-%20LinuxCon_0.pdf
- http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide
- http://www.solarisinternals.com/wiki/index.php/ZFS_Configuration_Guide
- How disks fail http://blog.backblaze.com/2013/11/12/how-long-do-disk-drives-last/
- Understanding the Robustness of SSDs under Power Fault https://www.usenix.org/system/files/conference/fast13/fast13-final80.pdf
- zxfer http://forums.freebsd.org/showthread.php?t=24113