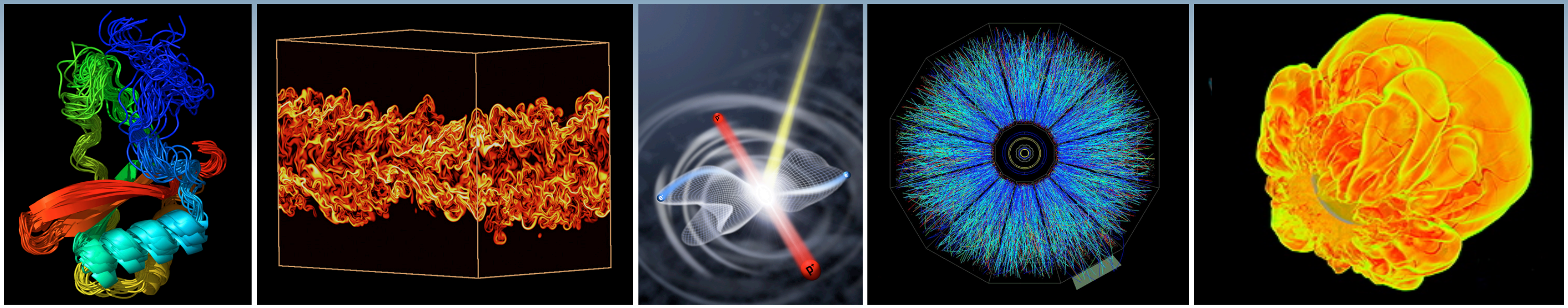


No-Cost ZFS
On Low-Cost Hardware

Trever Nightingale
Senior Systems Analyst
NERSC Server Team

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

Part One

Why Use ZFS?

Motivation

Task: New IMAP Server

- **several terabytes** of email highly available over long term
- near real time replica server standing by

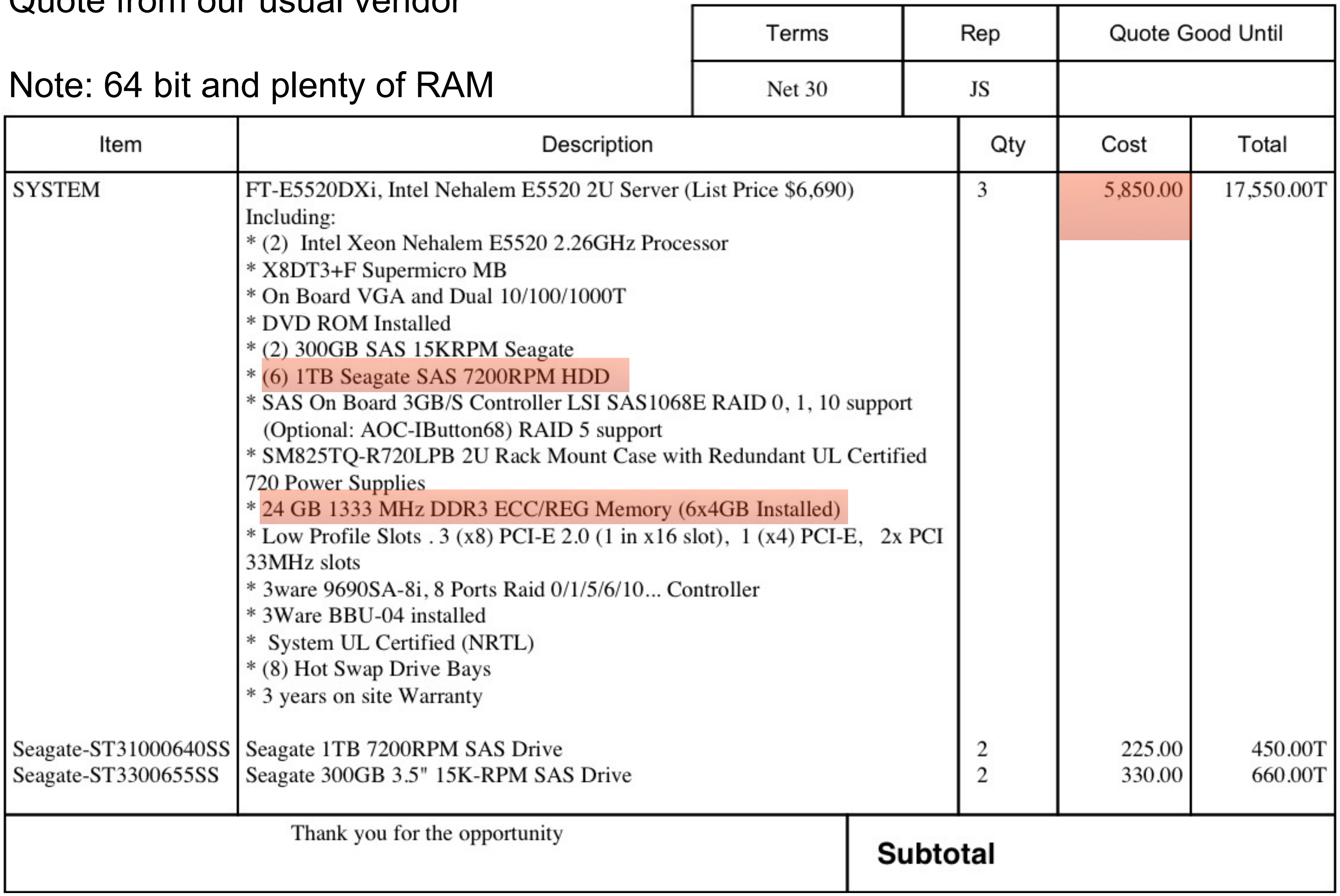
Tools budgeted for the task:

- 2U, 8 drive bay commodity "white box" hardware
- Open source no cost software

Order the commodity hardware

Quote from our usual vendor

Note: 64 bit and plenty of RAM

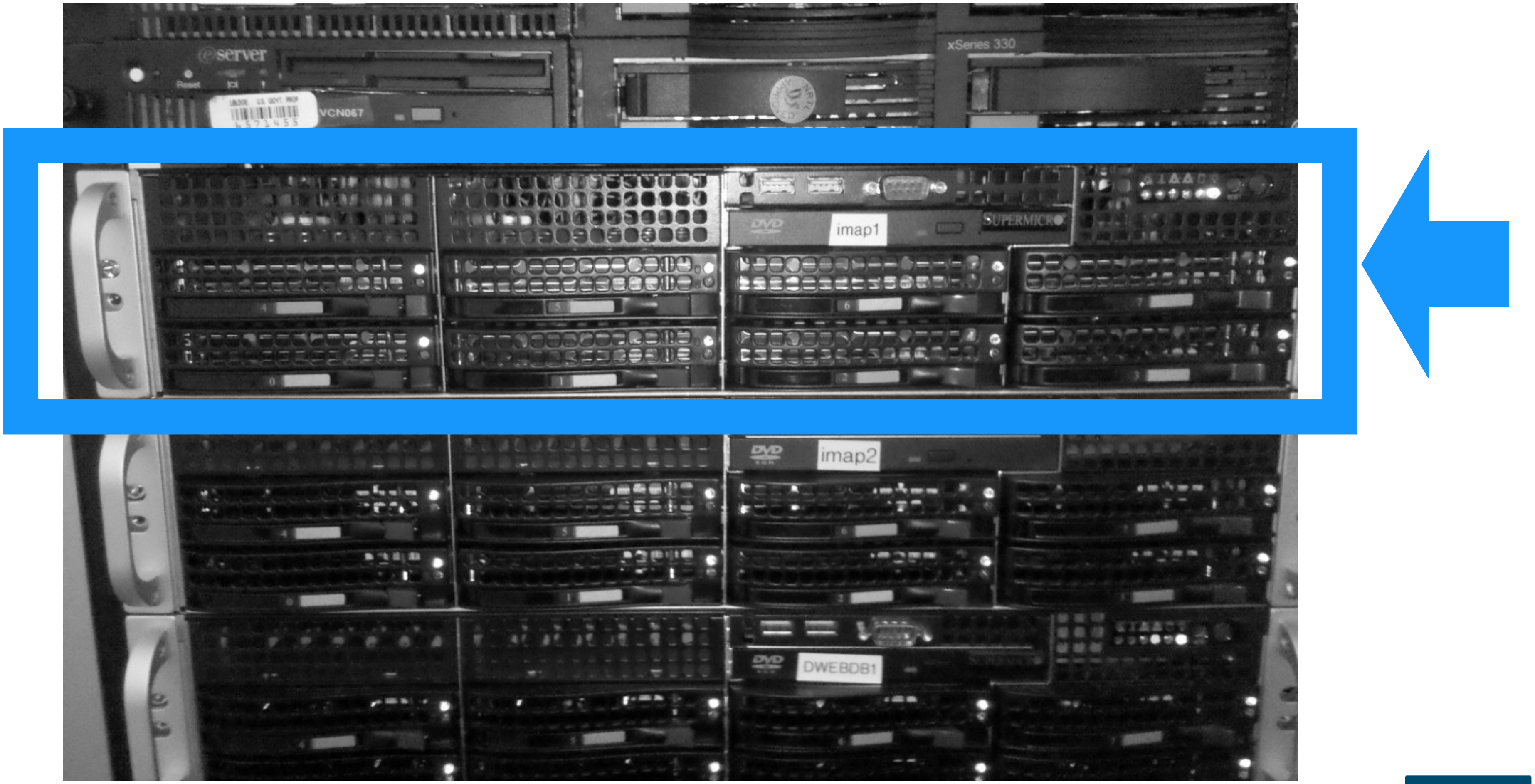


Terms	Rep	Quote Good Until
Net 30	JS	

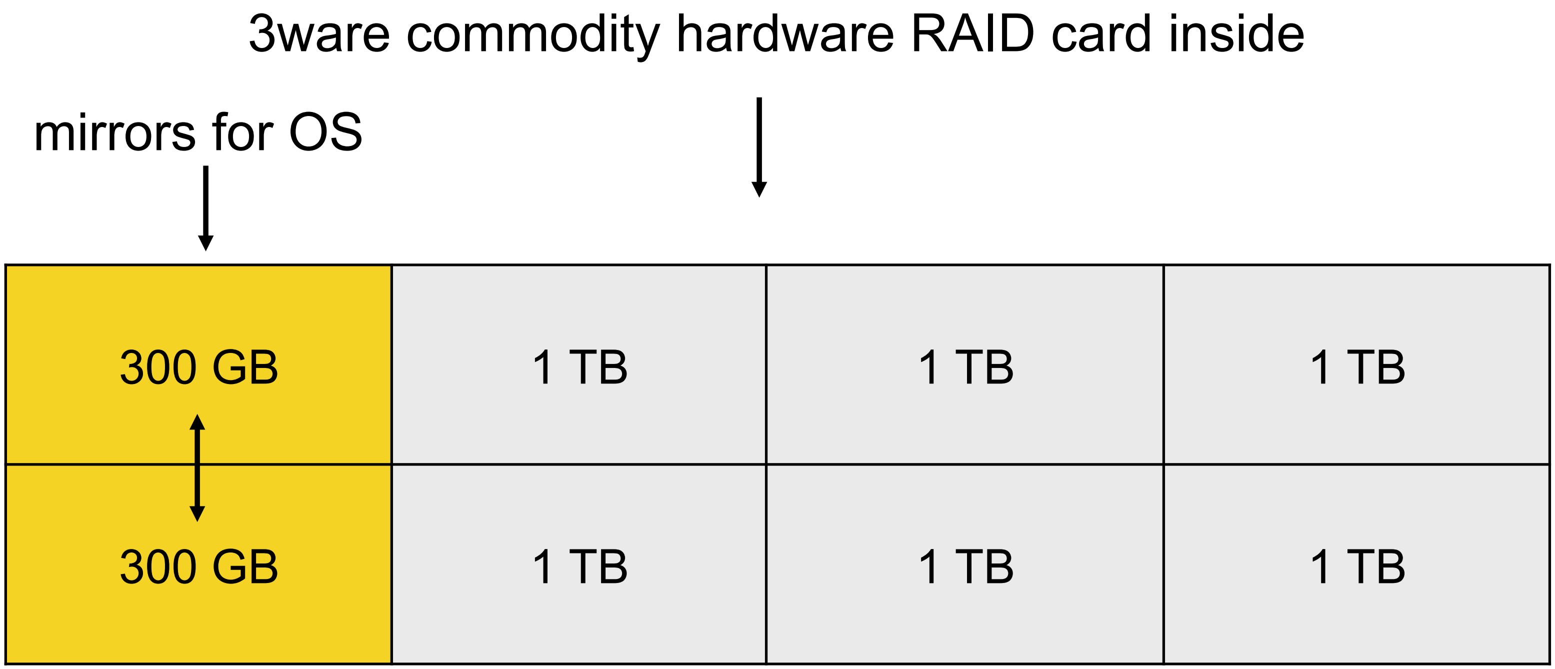
Item	Description	Qty	Cost	Total
SYSTEM	FT-E5520DXi, Intel Nehalem E5520 2U Server (List Price $6,690) Including: * (2) Intel Xeon Nehalem E5520 2.26GHz Processor * X8DT3+F Supermicro MB * On Board VGA and Dual 10/100/1000T * DVD ROM Installed * (2) 300GB SAS 15KRPM Seagate * (6) 1TB Seagate SAS 7200RPM HDD * SAS On Board 3GB/S Controller LSI SAS1068E RAID 0, 1, 10 support (Optional: AOC-IButton68) RAID 5 support * SM825TQ-R720LPB 2U Rack Mount Case with Redundant UL Certified 720 Power Supplies * 24 GB 1333 MHz DDR3 ECC/REG Memory (6x4GB Installed) * Low Profile Slots . 3 (x8) PCI-E 2.0 (1 in x16 slot), 1 (x4) PCI-E, 2x PCI 33MHz slots * 3ware 9690SA-8i, 8 Ports Raid 0/1/5/6/10... Controller * 3Ware BBU-04 installed * System UL Certified (NRTL) * (8) Hot Swap Drive Bays * 3 years on site Warranty	3	5,850.00	17,550.00T
Seagate-ST31000640SS	Seagate 1TB 7200RPM SAS Drive	2	225.00	450.00T
Seagate-ST3300655SS	Seagate 300GB 3.5" 15K-RPM SAS Drive	2	330.00	660.00T

Thank you for the opportunity	**Subtotal**

The hardware arrives



What you have...



3ware commodity hardware RAID card inside

mirrors for OS

300 GB	1 TB	1 TB	1 TB
300 GB	1 TB	1 TB	1 TB

What Do You Do With Those 6 x 1 TB Discs?

One possibility…

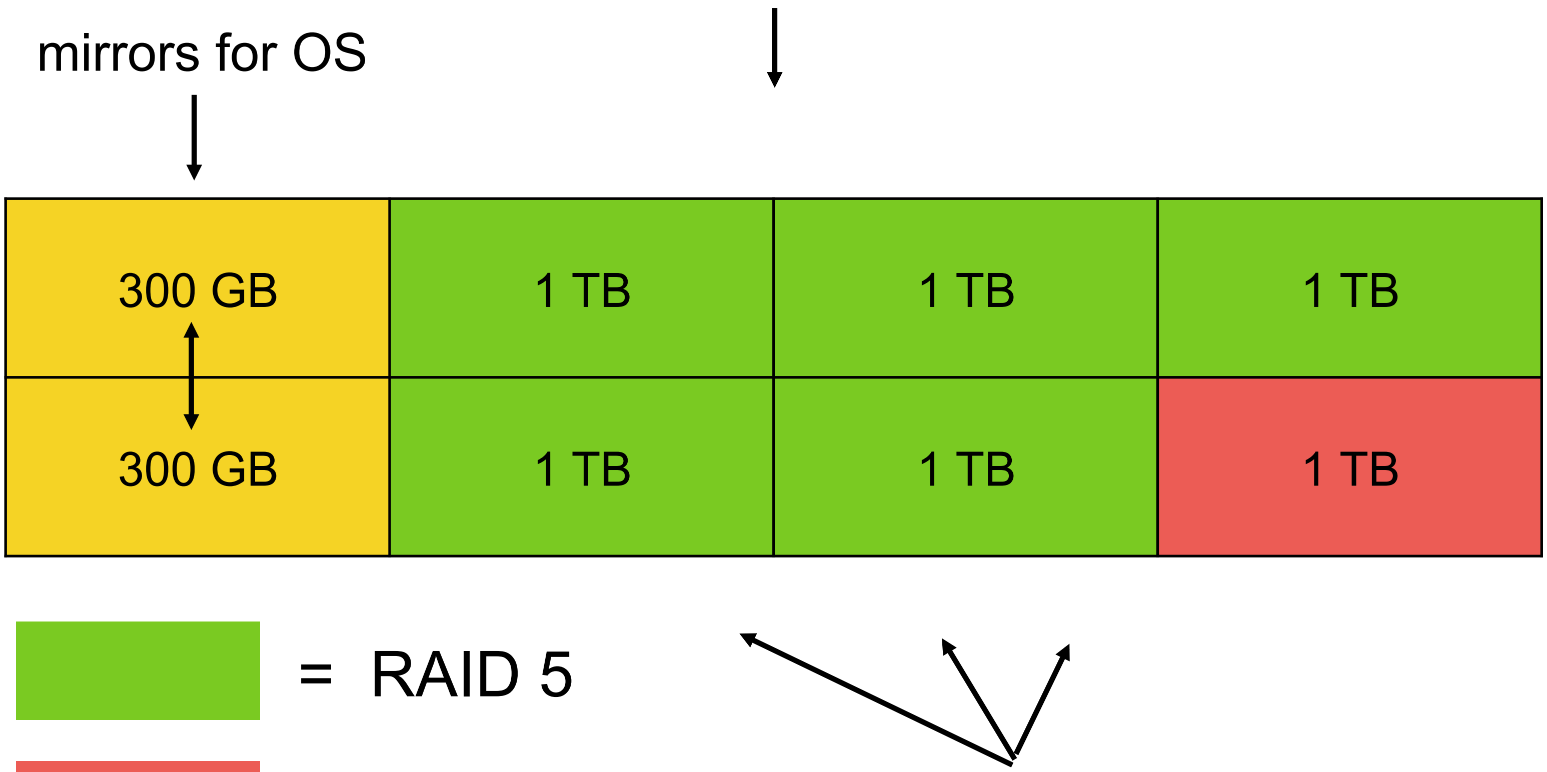
Try Business as usual

Hardware RAID all the drives

now you have...

3ware commodity hardware RAID card inside

mirrors for OS



300 GB	1 TB	1 TB	1 TB
300 GB	1 TB	1 TB	1 TB

🟩 = RAID 5

🟥 = Hot Spare

Email goes on RAID 5

This gives you ~4 TB LUN for your email filesystem

Note: all email must collect on **one** filesystem

So are we done?

Not yet, we need to put a filesystem on that ~4 TB LUN

Since this is business as usual, let's assume the 1 TB drives in our LUN behave like usual

Can we create a 4 TB filesystem with our free software?

Not if it's a FreeBSD UFS filesystem

- Michael Lucas, Absolute FreeBSD author: “In my opinion, soft updates are suitable for partitions of less than 80 GB or so.”
- Snapshots making systems unresponsive
- Max UFS filesystem size?

(Journaled soft updates for UFS not available yet)

Our existing IMAP was FreeBSD

Let's try switching to Linux

Business as usual on Linux is: ext3

Will ext3 scale to 4 TB?

Officially yes, but are we enthusiastic?

- My enthusiasm and confidence is not high. ext3 not optimally designed for filesystems this large
- We are approaching supported limits: 8 TB CentOS 4, 16TB CentOS 5

Filesystem Scaling Issues

For the boss's email, I'd like to feel more confident in my filesystem technology at 4 TB

What about those large 1 TB discs?

And what about our assumption about our hardware?

Do those discs behave like previous generation discs?

Trends in Storage Integrity

- Uncorrectable bit error rates have stayed roughly constant
  - 1 in 10^{14} bits (~12TB) for desktop-class drives
  - 1 in 10^{15} bits (~120TB) for enterprise-class drives (allegedly)
  - Bad sector every 8-20TB in practice (desktop and enterprise)
- Drive capacities doubling every 12-18 months
- Number of drives per deployment increasing
- → Rapid increase in error rates
- Both silent and “noisy” data corruption becoming more common
- Cheap flash storage will only accelerate this trend

Data integrity problems

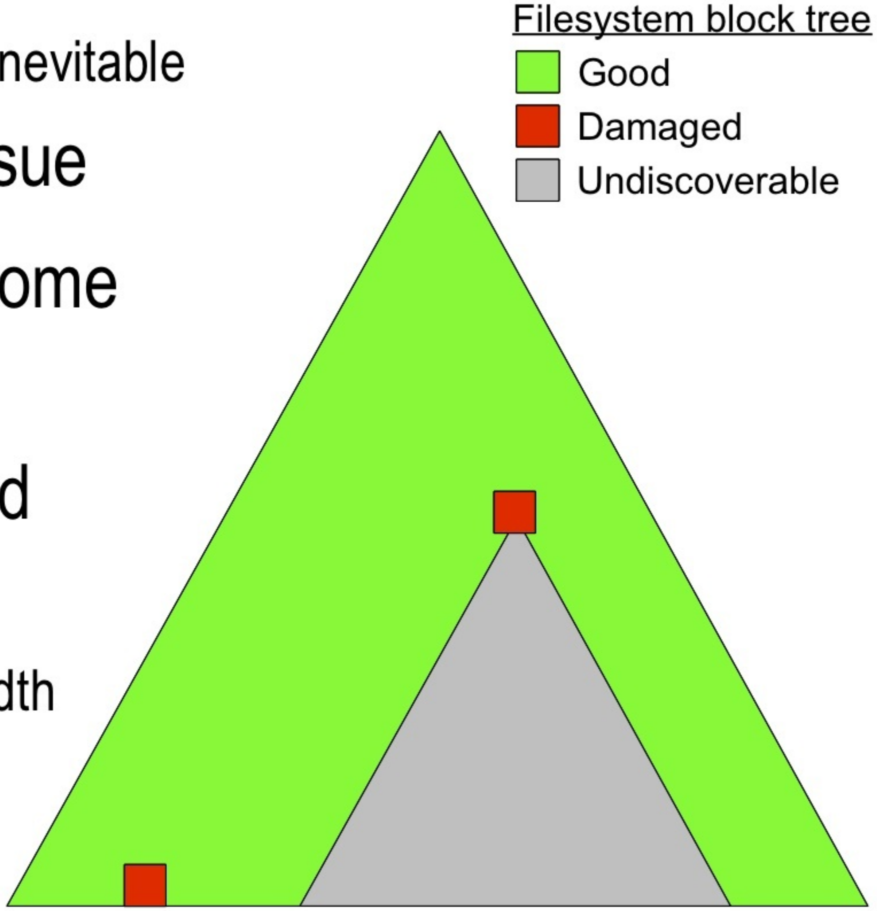
- Bit-rot: magnetic properties of media silently changed or damaged
- Bugs in drive firmware, RAID controller: misdirected writes and phantom writes
- Data transfer noise (UTP, SATA, FC)
- OS software bugs: drivers and filesystem code itself

Measurements at CERN

- Wrote a simple application to write/verify 1GB file
  - Write 1MB, sleep 1 second, etc. until 1GB has been written
  - Read 1MB, verify, sleep 1 second, etc.
- Ran on 3000 rack servers with HW RAID card
- After 3 weeks, found 152 instances of silent data corruption
  - Previously thought “everything was fine”
- HW RAID only detected “noisy” data errors
- Need end-to-end verification to catch silent data corruption

Surviving Multiple Data Failures

- With increasing error rates, multiple failures can exceed RAID's ability to recover
  - With a big enough data set, it's inevitable
- Silent errors compound the issue
- Filesystem block tree can become compromised
- “More important” blocks should be more highly replicated
  - Small cost in space and bandwidth



- Disc data integrity has always been at least a small problem

- New big drives means the disc data integrity problem is becoming more significant

Bigger drives behave differently…

…per drive error rate is different

A
300 GB

B
1 TB

More likely to get an error reading
B end-to-end than reading **A** end-to-end

Note: there is **no time** element here

Business as usual is looking problematic

- The problems with the newly available huge discs are not widely known and appreciated among sys admin circles (and beyond)

- It looks like we are arriving at someplace new on the technology curve

We Need A New Approach

- ZFS has end-to-end data integrity checking, well designed for protection against the potential errors with larger hard drives

- ZFS is free and production ready in FreeBSD

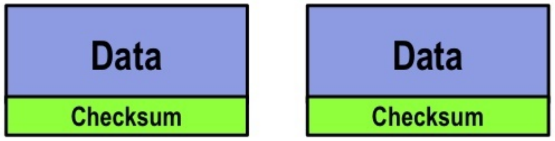
ZFS's Data Integrity Technology

- Maintains checksums for all on disc blocks
- Checksums are kept separate from corresponding blocks
- Checksums stored in a block's pointer structure (except uberblocks which have no parent ptrs)
- Before using a block, ZFS calculates its checksum and verifies it against the stored checksum in pointer

ZFS End to End Checksums

Disk Block Checksums

- Checksum stored with data block
- Any self-consistent block will pass
- Can't even detect stray writes
- Inherent FS/volume interface limitation

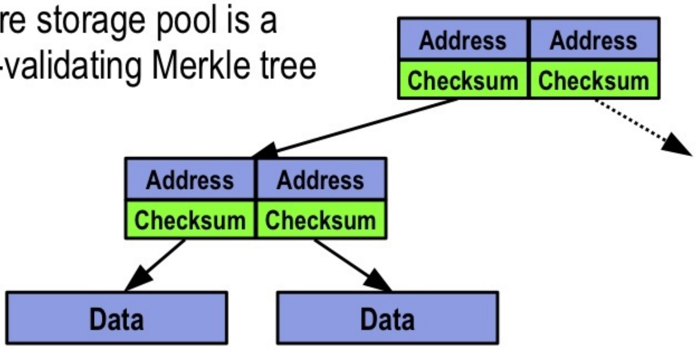


Disk checksum only validates media

- ✓ Bit rot
- ✗ Phantom writes
- ✗ Misdirected reads and writes
- ✗ DMA parity errors
- ✗ Driver bugs
- ✗ Accidental overwrite

ZFS Data Authentication

- Checksum stored in parent block pointer
- Fault isolation between data and checksum
- Entire storage pool is a self-validating Merkle tree



ZFS validates the entire I/O path

- ✓ Bit rot
- ✓ Phantom writes
- ✓ Misdirected reads and writes
- ✓ DMA parity errors
- ✓ Driver bugs
- ✓ Accidental overwrite

ZFS detected & endured this

- Flipped bits at random offsets in 9 different classes of disc blocks using a pseudo-driver interposed between ZFS virtual device and disc driver

- Corrupted metadata blocks, then did mounts of unmounted and remounts of mounted filesystems

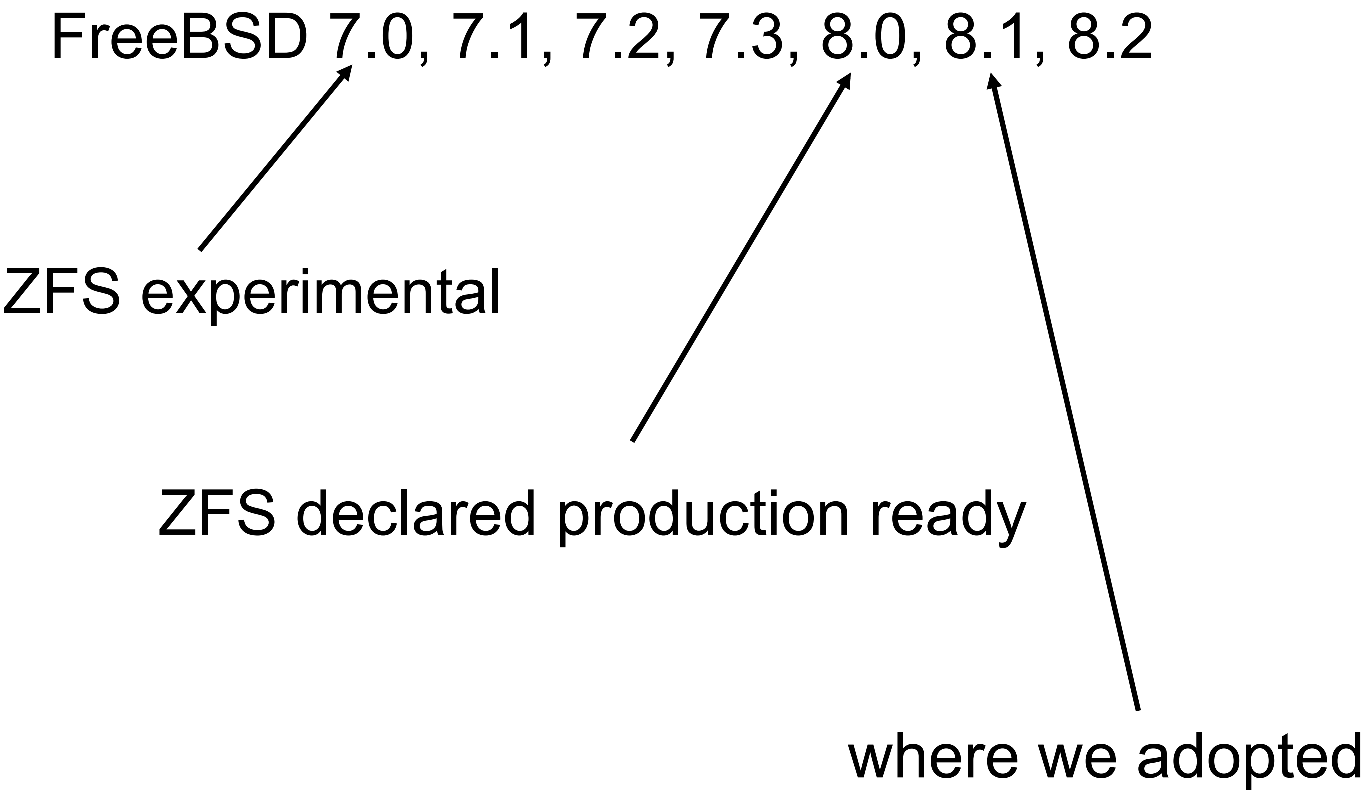
- Corrupted data and directory blocks, did a read file or a create file in a directory

See “End-to-end Data Integrity for File Systems: A ZFS Case Study”, University of Wisconsin-Madison

ZFS Strengths

- ZFS demonstrated to protect against the data integrity problems of huge discs
- ZFS is *designed* for large filesystems (it’s 128 bit): Maximum filesystem size is 256 quadrillion zettabytes (1 zettabyte = 2^{70})
- ZFS code base itself is well regarded and in production ~6 years (in Solaris since June 2006)

ZFS port to FreeBSD declared production ready



FreeBSD 7.0, 7.1, 7.2, 7.3, 8.0, 8.1, 8.2

ZFS experimental

ZFS declared production ready

where we adopted

Why ZFS?

Motivation

It looks riskier to not try ZFS than to try it

Argument Summary:

1. Proven/designed to scale to large filesystems
2. On commodity hardware, you need ZFS's data integrity at today's disc sizes
3. tomorrow's disc sizes increase the data integrity need

Part Two

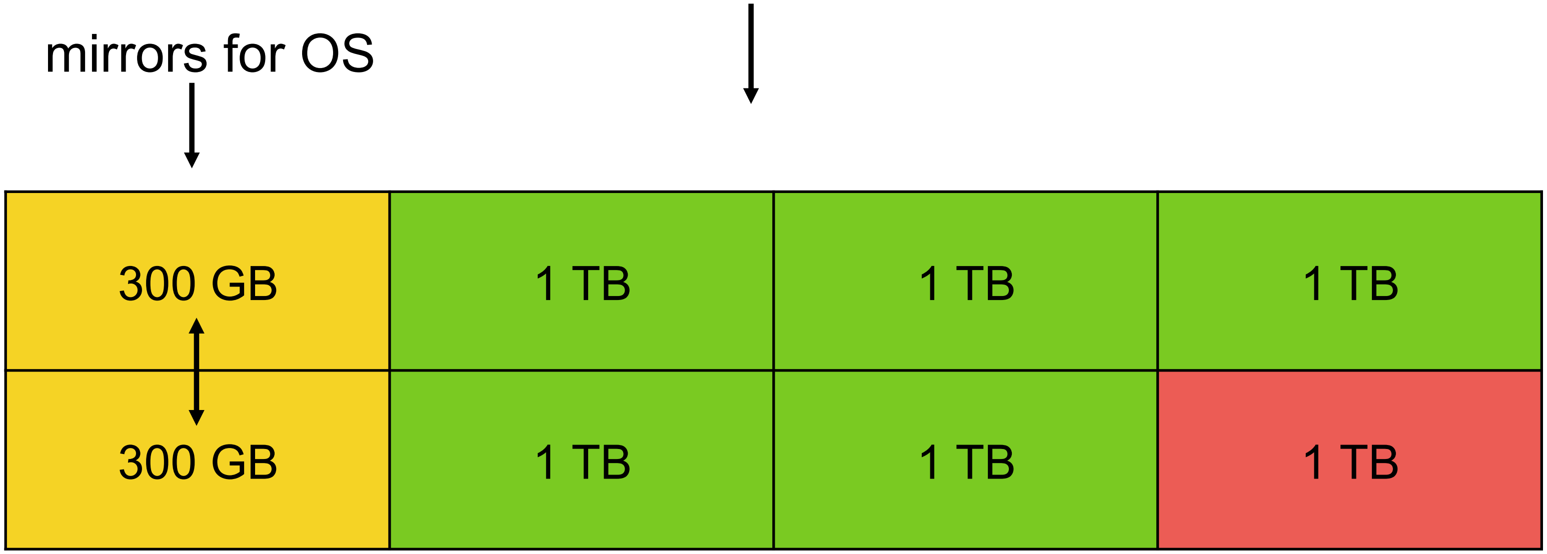
Deploying ZFS

How did it go?

now you have...

3ware commodity hardware RAID card inside

mirrors for OS



300 GB	1 TB	1 TB	1 TB
300 GB	1 TB	1 TB	1 TB

🟩 = RAIDZ

🟥 = Hot Spare

Email goes on RAIDZ

RAIDZ is not hardware RAID

- 3ware RAID card only mirroring the two OS drives
- 3ware RAID card is just exporting the six remaining 1 TB drives to the OS (JBOD)

RAIDZ is built into ZFS

- ZFS prefers raw discs to do its magic

- RAIDZ has no RAID5 write hole

- Resilvering uses checksums (does not blindly copy blocks underneath other layers)

- Software RAID that is actually preferable for ease of administration compared to hardware RAID! (Software RAID can be complex)

RAID5 Write Hole

Lose power after writing a data block but before writing the corresponding parity block → data and parity for that stripe are inconsistent

Result: RAID reconstruction in event of disc failure will generate garbage (silently)

ZFS RAIDZ Solution to write hole

Every write is a full-stripe write (no partial writes).
Combined with COW transactions →
no write hole

(Entire RAIDZ implementation: 599 lines of code)

Installation & Configuration

```
su -
zpool create -O compression=lzjb z raidz da1 da2 da3 da4 da5 spare da6
```

DONE

```
imap1 # df —hT

Filesystem        Type      Size     Used    Avail    Capacity    Mounted on
/dev/da0s1a       ufs       262G     4.1G     237G          2%    /
z                 zfs       3.5T      31k     3.5T          0%    /z
```

That's It!

Though it took awhile to believe it…

…and it will take awhile to get used to things

Part Three

ZFS Features

Yes it's love

ZFS integration

- Notice how everything was done for you
  no partitioning, labeling, fstab, newfs, creating volumes etc.

- No sizing or preallocation (dynamic allocation)

- Two commands: zfs and zpool

Transactional

- Always consistent on disc (COW)
- Not a journaling filesystem
- No fsck (and they refuse to create one)

Separate ZIL (ZFS Intent Log)

- Not a journal for consistency:
  ZIL log replay is NOT about re-establishing consistency, unlike journaled filesystems. ZFS pools always come up in a consistent state, but any ZIL records can be incorporated into a new consistent state via replay.

- Supports synchronous write semantics separately from rest of I/O pipeline which allows for optimized overall and synchronous performance (database servers, NFS servers)

- Can easily be put on SSD or low latency media, or separate spindles in your pool

Commodity Database server

3ware commodity hardware RAID card inside for OS mirror, rest is ZFS

🟩 2 TB	🟩 2 TB	🟩 2 TB	🟩 2 TB
🟩 2 TB	🟩 2 TB	🟩 2 TB	🟩 2 TB
🟥 2 TB	🟩 2 TB	🟩 2 TB	🟩 2 TB
🟥 2 TB	🟩 2 TB	🟩 2 TB	🟩 2 TB
🟨 300 GB	🟩 2 TB	🟩 2 TB	🟩 2 TB
🟨 300 GB	🟩 2 TB	🟩 2 TB	🟩 2 TB

🟨 = OS mirror

🟥 = ZFS mirror for ZIL

🟩 = RAIDZ2

Everything is fast

- Pool creation, filesystem creation are instantaneous!
- Makes heavy use of memory (ARC) and state of the art in filesystem tech for performance

OMG, snapshots!

- Free. Absolutely zero performance impact
- 2^{64} per filesystem
  (by comparison, UFS max. is 20)
- zfs rollback (*undo command for your servers!*)
- Tape is now truly only for a total disaster (good riddance to 99% of all tape restores)

Snapshots in action

"portland"

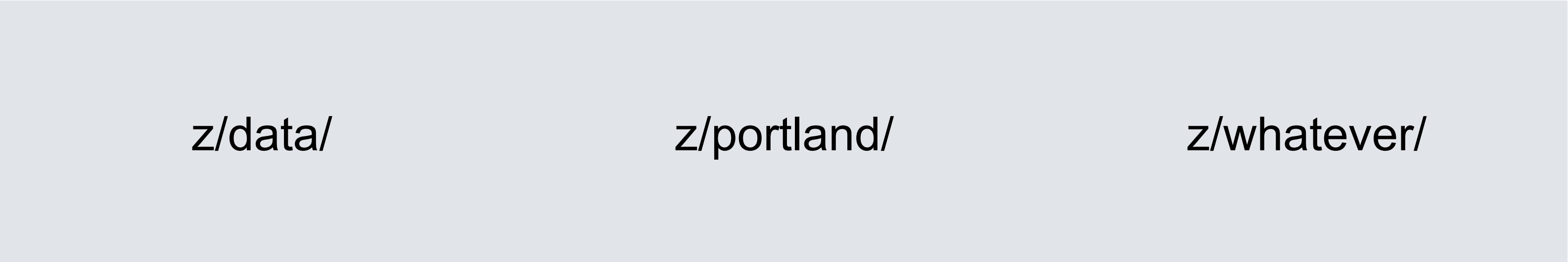
```
imap1 # df -hT

Filesystem          Type      Size      Used     Avail     Capacity    Mounted
on

/dev/da0s1a         ufs       262G      4.1G      237G           2%    /

z                   zfs       3.5T       31k      3.5T           0%    /z
```

ZFS filesystem hierarchy
(or namespace)

"Z space"

System filesystem hierarchy
(or namespace)

Within the ZFS heirarchy, all filesystems go under the root filesystem

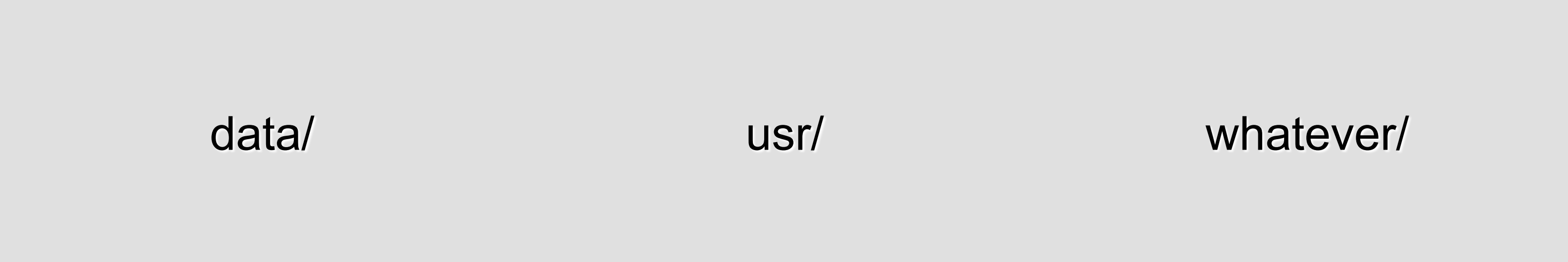


single ZFS pool

All go under root in "Z space", but can mount anywhere in filesystem:

```
z/portland/usr/ports           /usr/ports
z/data                         /data
```

You can **not** do this:



single ZFS pool

You can **not** have this:

```
usr/ports                          /usr/ports
data                               /data
```

Create A Portland Tree

z/portland/ z/data/ z/whatever/

z/portland/usr/local

z/portland/usr/ports

z/portland/usr/ports/

z/portland/var/db/pkg

z/portland/var/db/ports

single ZFS pool

Create "portland”

```
export Z=“zfs create –p –o compression=lzj”
$Z z/portland/usr/local
$Z z/portland/usr/ports/distfiles
$Z z/portland/var/db/pkg
$Z z/portland/var/db/ports
$Z z/portland/var/db/portsnap
mkdir /usr/ports
zfs set mountpoint=/usr/local z/portland/usr/local
zfs set mountpoint=/usr/ports z/portland/usr/ports
zfs set mountpoint=/var/db/ports z/portland/var/db/ports
zfs set mountpoint=/var/db/portsnap z/portland/var/db/portsnap
zfs set mountpoint=/var/db/pkg z/portland/var/db/pkg
```

(2^{64} filesystems per pool)

Portland

```
imap1# df –hT | grep portland
z/portland/usr/local                    /usr/local
z/portland/usr/ports                    /usr/ports
z/portland/usr/ports/distfiles    /usr/ports/distfiles
z/portland/var/db/pkg                   /var/db/pkg
z/portland/var/db/ports                 /var/db/ports
z/portland/var/db/portsnap        /var/db/portsnap
z/portland                              /z/portland
z/portland/usr                          /z/portland/usr
z/portland/var                          /z/portland/var
z/portland/var/db                       /z/portland/var/db
```

Portland in action

```
zfs snapshot -r z/portland@base_install
```

Upgrade software stack. Upgrade breaks LDAP. Do:

```
zfs rollback -r z/portland@base_install
```

- atomic down the entire heirarchy (-r)
- Portland will likely stay even with an all ZFS system (even with ZFS root)

ZFS State of the art

- 2^{48} — Number of entries in any individual directory
- 16 EB — Maximum size of a single file
- 16 EB — Maximum size of any attribute
- 256 ZB (2^{78} bytes) — Maximum size of any zpool
- 2^{56} — Number of attributes of a file (constrained to 2^{48} for the number of files in a ZFS file system)
- 2^{64} — Number of devices in any zpool
- 2^{64} — Number of zpools in a system
- 2^{64} — Number of file systems in a zpool

Remote replication

Fully score boarded I/O pipeline

Aggregation

Deadline scheduling

DeDuplication

Intelligent Prefetch

Dynamic Striping

Snapshots

Clones

Compression

Adaptive replacement cache

128 bit

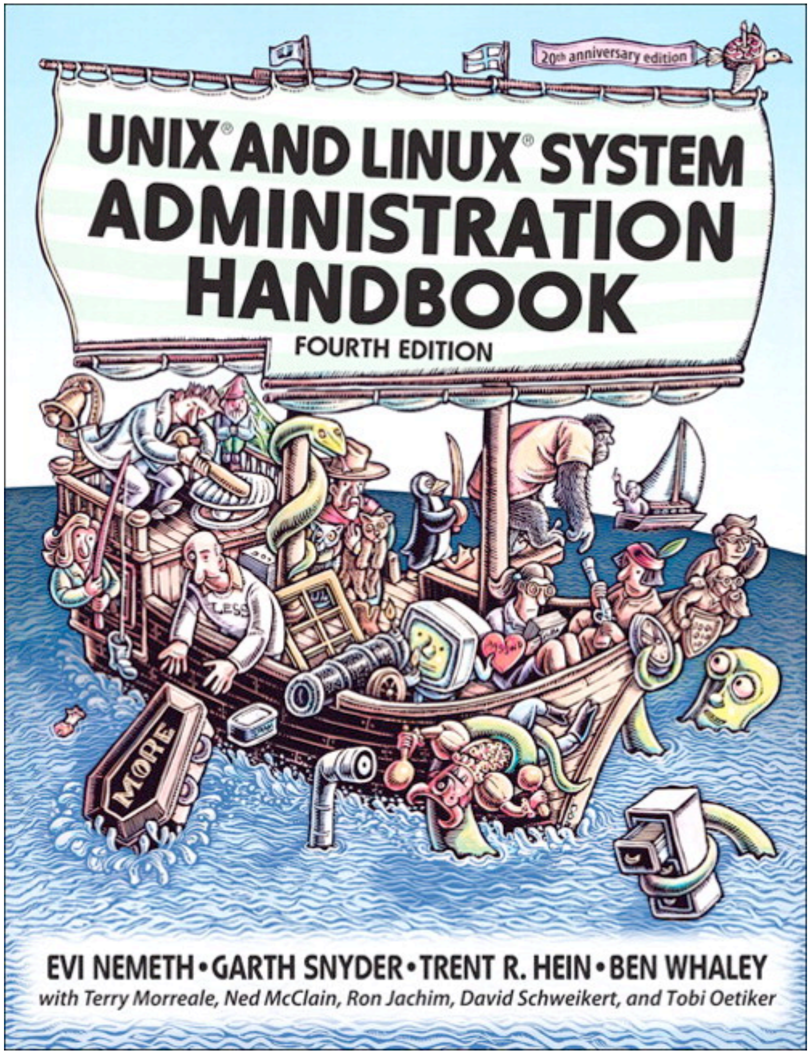
Ditto blocks

Transactional

What they say about ZFS

"All your storage problems solved"

"... we predict you will enjoy working with it. There is little doubt that the system will be widely emulated over the next decade. The open question is how long we'll have to wait to get ZFS-style features on other systems."


20th anniversary edition
UNIX® AND LINUX® SYSTEM ADMINISTRATION HANDBOOK
FOURTH EDITION
MORE
LESS
EVI NEMETH • GARTH SNYDER • TRENT R. HEIN • BEN WHALEY
with Terry Morreale, Ned McClain, Ron Jachim, David Schweikert, and Tobi Oetiker

One more thing...

That 3.5 TB filesystem?

It holds 7 TB of email (twice as much!)

ZFS has built in compression
Faster with compression (cpu is faster than disc)

7 TB of highly reliable disc space in a standard (2 U)
State of the art performance
All for the low price of $6k

Part Four

ZFS Lessons Learned

ZFS send/receive

ZFS's native ability to serialize the filesystem
Pipe filesystem from one place to another

Two hopes for NERSC server team:

1) Backup
2) remote mirroring (IMAP standby)

Backing up ZFS at NERSC

Outline of logic for a “full” backup

```
for filesystem in `zfs list -H -r -o name`
do

filesystem="$filesystem@00daysago”

/usr/bin/ftp hpss

put "| zfs send $filesystem | gzip -9” $HPSSFILENAME

done
```

Backing up ZFS at NERSC

- can have full (with or without history- i.e. snapshots) and incrementals (but snapshots are already on disc)
- more filesystems = greater granularity for backups + smaller backup files lowers chance of corrupt backup files
- zfs send is verifying checksums as it reads
- SUN/Oracle: zfs send not a backup solution (they say this because can’t restore individual files)
- NERSC: ZFS native backups just as good as UFS dump

ZFS send/receive

Backups went well, work on par with rest of our backups

How about mirroring?

IMAP mirroring requirements

- Create a standby mirror IMAP server
- Mirroring can be non-realtime
- Some data loss acceptable

Eg. sync every 30 minutes

ZFS send/receive

```
zfs send | ssh mirror zfs receive
```

- When incremental updates are interrupted, must re-initialize mirror from beginning
  (i.e. loss of mirror until re-initialization complete)

  Bug? Not doing correctly? Unanswered

- SLOW
  40 GB taking ~10 minutes for incremental

rsync was the better solution

Caveats…unlike ZFS send/receive

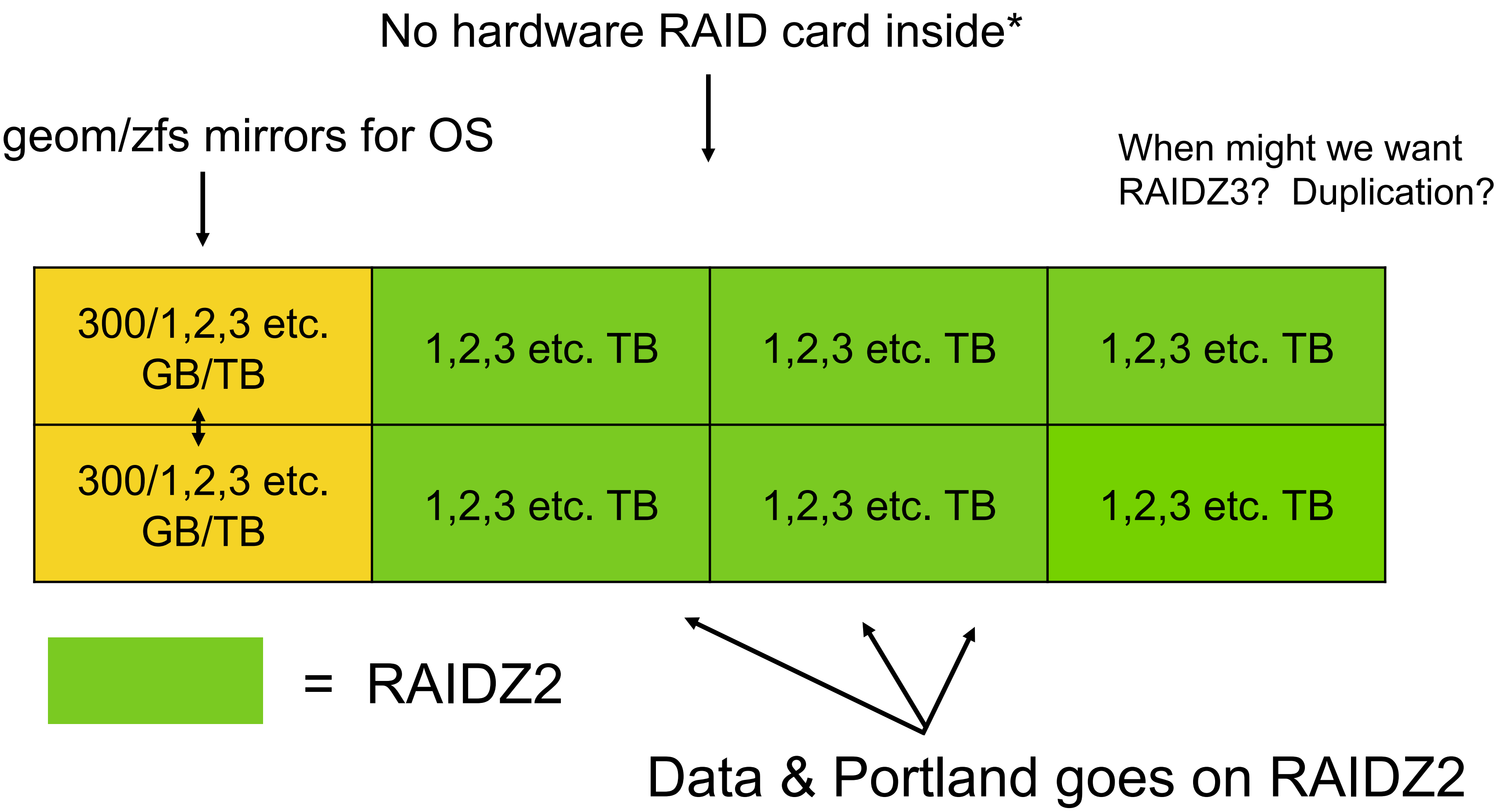
- Mirror has it’s own history
- Filesystem additions/deletions and structural changes not propagated

ZFS send/receive

Remote mirroring disappointing

(pace blog/web)

Going forward…



No hardware RAID card inside*

geom/zfs mirrors for OS

When might we want RAIDZ3? Duplication?

300/1,2,3 etc. GB/TB	1,2,3 etc. TB	1,2,3 etc. TB	1,2,3 etc. TB
300/1,2,3 etc. GB/TB	1,2,3 etc. TB	1,2,3 etc. TB	1,2,3 etc. TB

🟩 = RAIDZ2

Data & Portland goes on RAIDZ2

* At least two reasons (driver/zfs.ko problems, what is it doing?)

Going forward…

Did we get the commodity hardware right?

Eg. always honors write barrier?

“Catastrophically destroyed pool”
Jeff Bonwick, Sun Fellow, ZFS team lead

Avoid

Today at NERSC, ZFS in production

- imap1 & 2 → 6 months online
- new ldap infrastructure → 1.5 months online master, 3 replicas

No incidents to date (5 servers)

12 more on deck!

- User DB servers (2)
- Central logging and analysis servers (3)
- Login servers (2)
- Mailman server (1)
- Mail exchangers (2)
- NERSC internal DB servers (2)

References

1. http://en.wikipedia.org/wiki/ZFS

2. ZFS – The Last Word In Filesystems, Jeff Bonwick, Bill Moore, http://hub.opensolaris.org/bin/download/Community+Group+zfs/docs/zfslast.pdf

3. End-to-end Data Integrity for File Systems: A ZFS Case Study, Yupu Zhang, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Computer Sciences Department, University of Wisconsin-Madison, http://www.cs.wisc.edu/wind/Publications/zfs-corruption-fast10.pdf

4. ZFS On-Disk Specification, http://hub.opensolaris.org/bin/download/Community+Group+zfs/docs/ondiskformat0822.pdf

5. ZFS Source Tour, http://hub.opensolaris.org/bin/view/Community+Group+zfs/source

6. ZFS: The Next Word, Jeff Bonwick, Bill Moore, http://www.youtube.com/watch?v=Spd5qwGz35k

References

7. ZFS in the Trenches, Ben Rockwood, http://wikis.sun.com/download/attachments/63226450/ZFSintheTrenches.pdf

8. Jeff Bonwick's blog, feed://blogs.oracle.com/bonwick/en_US/feed/entries/rss?cat=%2FZFS

9. ZFS, copies, and data protection, Richard Elling's Weblog, http://blogs.oracle.com/relling/entry/zfs_copies_and_data_protection

10. Why RAID 5 stops working in 2009, Robin Harris, http://www.zdnet.com/blog/storage/why-raid-5-stops-working-in-2009/162

11. An Analysis of Data Corruption in the Storage Stack, Lakshmi N. Bairavasundaram∗ , Garth R. Goodson†, Bianca Schroeder‡ Andrea C. Arpaci-Dusseau∗ , Remzi H. Arpaci-Dusseau∗ , ∗ University of Wisconsin-Madison † Network Appliance, Inc. ‡ University of Toronto {laksh, dusseau, remzi}@cs.wisc.edu, garth.goodson@netapp.com, bianca@cs.toronto.edu, http://www.cs.toronto.edu/~bianca/papers/fast08.pdf

12. Triple-Parity RAID and Beyond, ADAM LEVENTHAL, http://portal.acm.org/ft_gateway.cfm?id=1670144&type=pdf