



EMPOWER: YOU

ZFS: Revolution in File Systems

Roman Strobl, OpenSolaris Evangelist

<http://blogs.sun.com/observatory>

SUN TECH DAYS 2008–2009
A Worldwide Developer Conference

Agenda

- **What is ZFS and why ZFS?**
- Volumes vs. pooled storage
- ZFS transactions & snapshots
- ZFS administration
- Conclusion



ZFS Overview

- ZFS = ZettaByte Filesystem
 - > Zetta = 10^{21}
- Pooled storage
 - > Completely eliminates notion of volumes
- Transactional object system
 - > Always consistent on disk – no fsck, ever
- Provable end-to-end data integrity
 - > Detects and corrects silent data corruption
- Simple administration
 - > Concisely express your intent



Trouble with Existing Filesystems

- No defense against silent data corruption
 - > Any defect in disk, controller, cable, driver, laser, or firmware can corrupt data silently
- Brutal to manage
 - > Labels, partitions, volumes, provisioning, grow/shrink, /etc files...
 - > Lots of limits: filesystem/volume size, file size, number of files, files per directory, number of snapshots ...
- Different tools to manage file, block, iSCSI, NFS, ...
- Not portable between platforms
- Dog slow
 - > Linear-time create, fat locks, fixed block size, naïve prefetch, dirty region logging, painful RAID rebuilds, growing backup time

ZFS Objective

End the suffering

- Figure out why storage has gotten so complicated
- Blow away 20 years of obsolete assumptions
- Design an integrated system from scratch

Agenda

- What is ZFS and why ZFS?
- **Volumes vs. pooled storage**
- ZFS transactions & snapshots
- End-to-end integrity
- ZFS administration
- Conclusion



Why Volumes Exist?

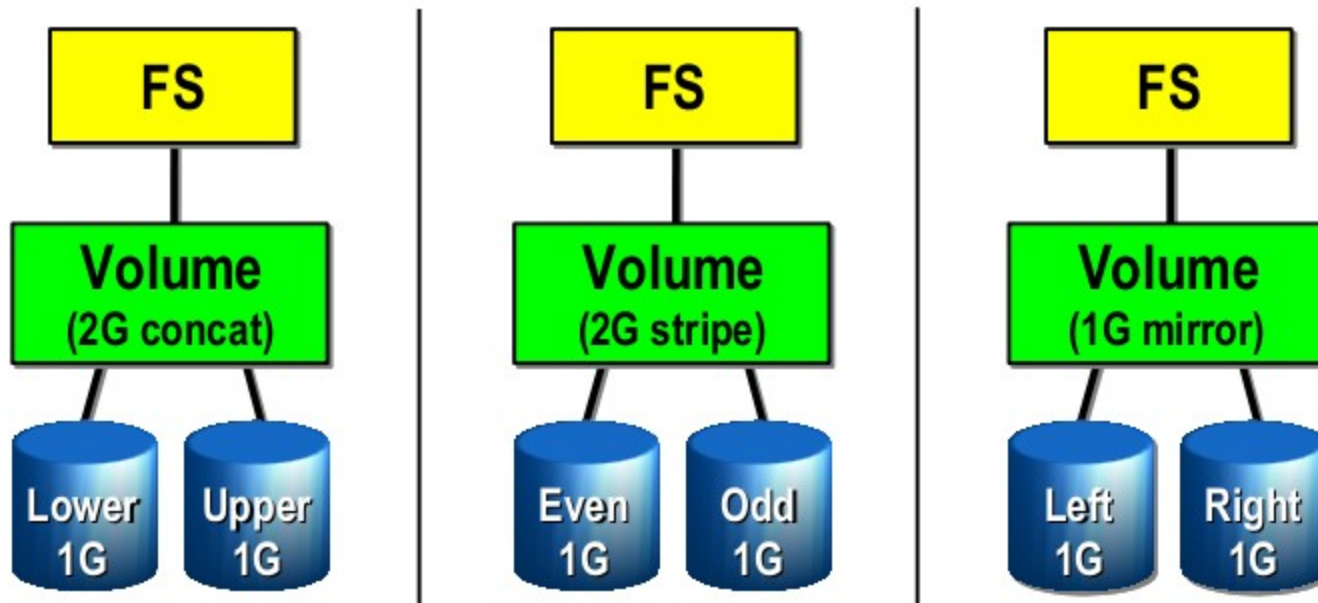
- Managing filesystems used to be simple





Why Volumes Exist? (continued)

- Need for concat, striping, mirrors, etc.



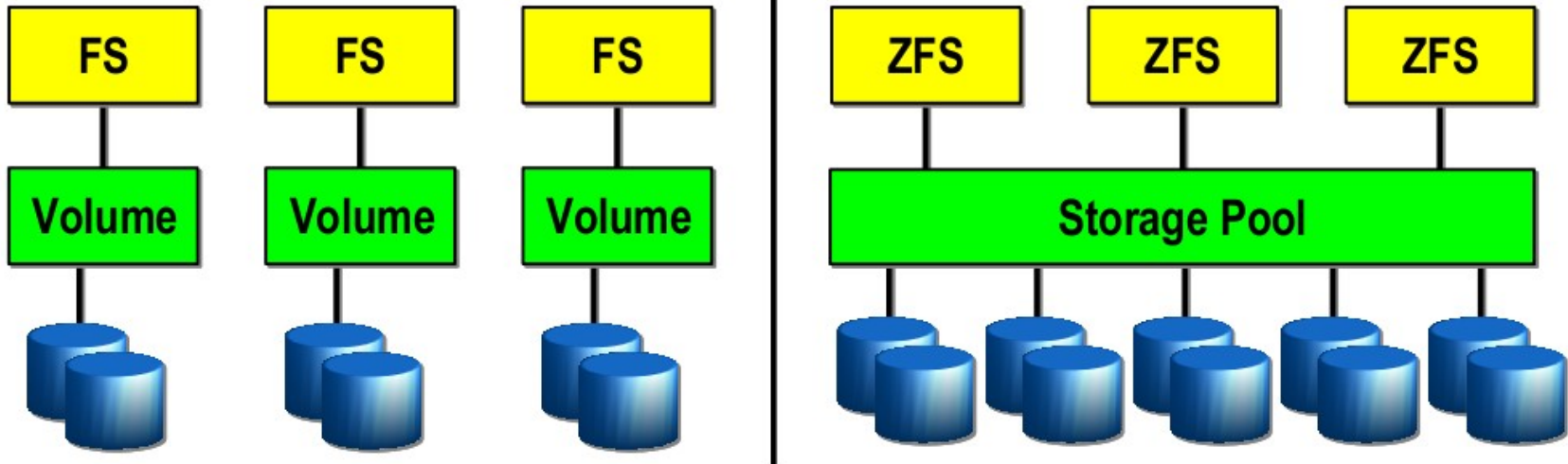


Issues with Volumes

- Each FS has limited storage/bandwidth
 - > Can't use disks used by other volumes
 - > Can't utilize full bandwidth of all disks
- Hard to grow/shrink
- Storage is fragmented
- Need for volume managers
 - > Another tool to learn
 - > Managing volumes takes extra time



Solution: Pooled Storage



- Advantages:
 - > All free storage space is always available
 - > Easy to grow/shrink
 - > No partitions to manage
 - > Abstraction: malloc/free



Demo: Pooled Storage

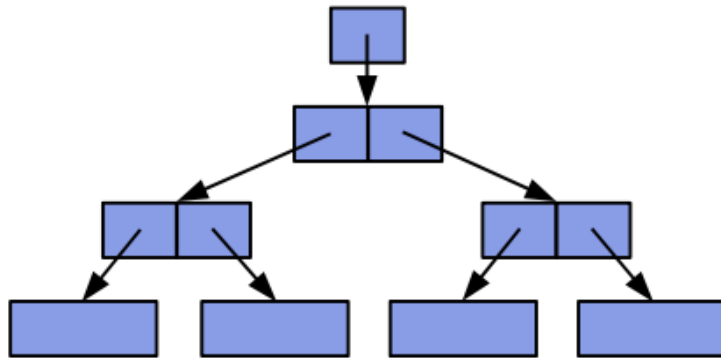
Agenda

- What is ZFS and why ZFS?
- Volumes vs. pooled storage
- **ZFS transactions & snapshots**
- ZFS administration
- Conclusion

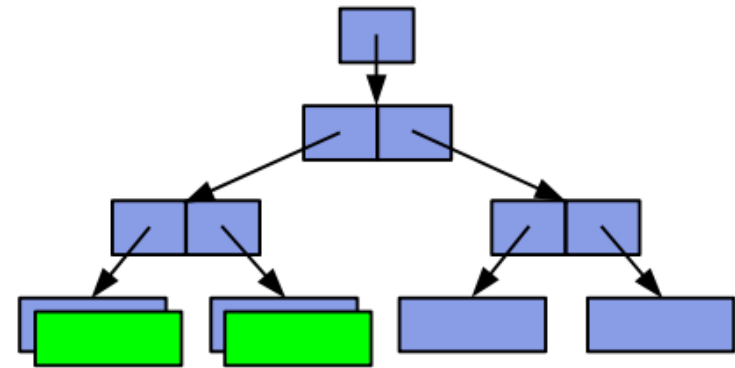


ZFS Transactions – Copy on Write

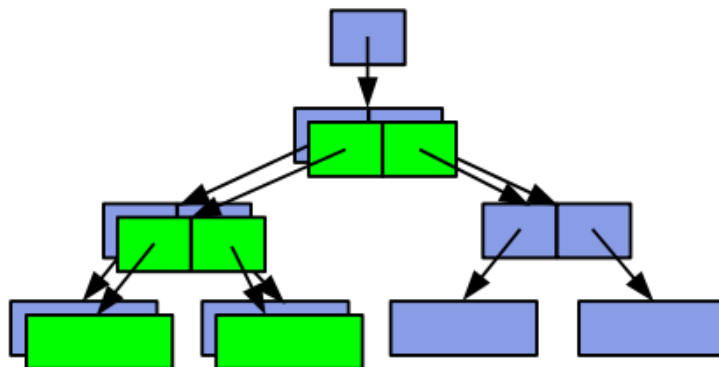
1. Initial block tree



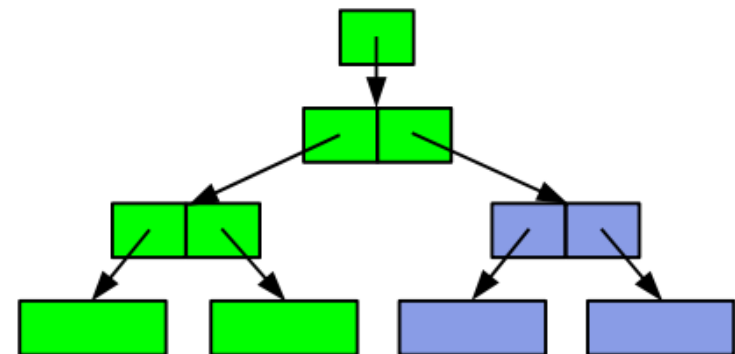
2. COW some blocks



3. COW indirect blocks



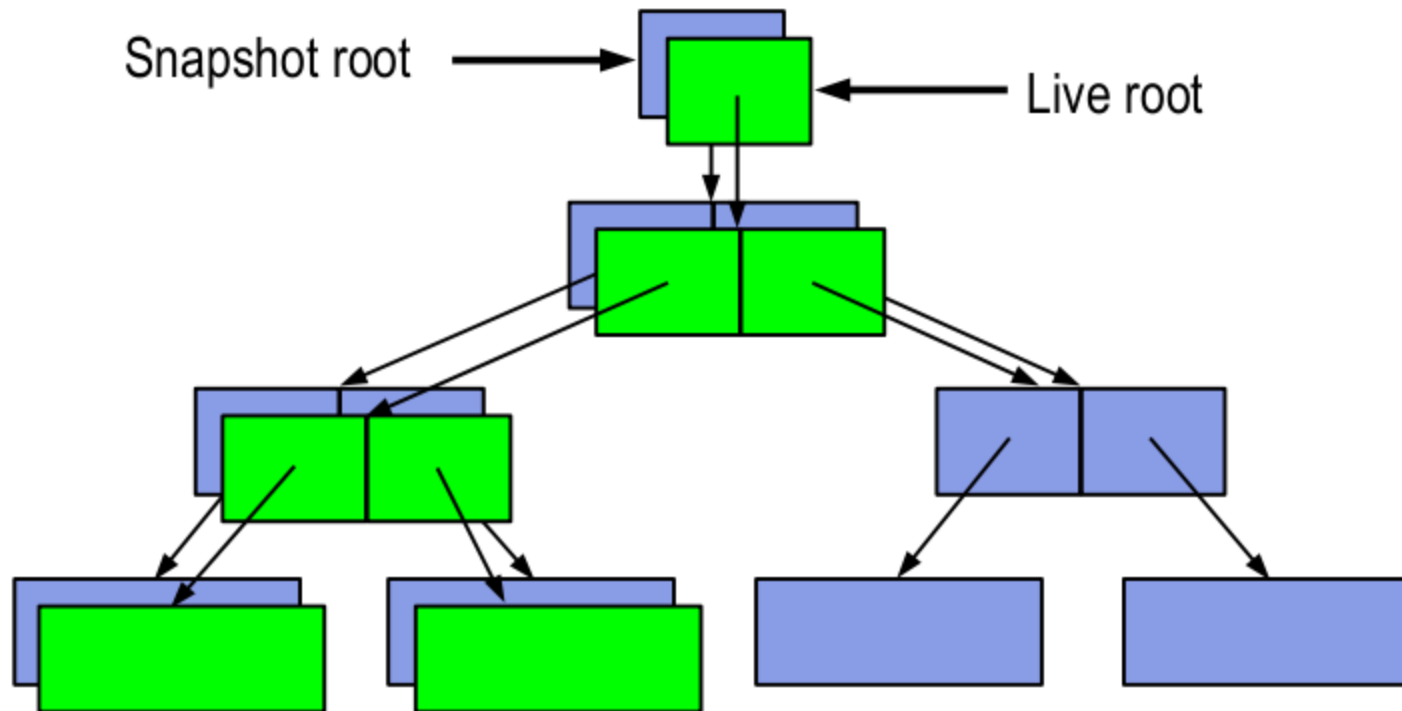
4. Rewrite uberblock (atomic)





ZFS Snapshots

- At the end of transaction, keep the old tree





Demo: ZFS Snapshots

Agenda

- What is ZFS and why ZFS?
- Volumes vs. pooled storage
- ZFS transactions & snapshots
- **ZFS administration**
- Conclusion



Pool and Filesystem Management

- Create a storage pool named “data”

```
# zpool create data mirror c0t0d0 c1t0d0
```

- Create home directory filesystem, mounted at /export/home

```
# zfs create data/home  
# zfs set mountpoint=/export/home data/home
```

- Create home directories for several users

Note: automatically mounted at /export/home/{ann,bob,sue} thanks to inheritance

```
# zfs create data/home/ann  
# zfs create data/home/bob  
# zfs create data/home/sue
```

- Later, add more space to the pool

```
# zpool add data mirror c2t0d0 c3t0d0
```



Setting Properties

- Automatically NFS-export all home directories

```
# zfs set sharenfs=rw tank/home
```

- Turn on compression for everything in the pool

```
# zfs set compression=on tank
```

- Limit Eric to a quota of 10g

```
# zfs set quota=10g tank/home/eric
```

- Guarantee John a reservation of 20g

```
# zfs set reservation=20g tank/home/john
```



ZFS Snapshots

- Read-only point-in-time copy of a filesystem
 - Instantaneous creation, unlimited number
 - No additional space used – blocks copied only when they change
 - Accessible through `.zfs/snapshot` in root of each filesystem
- Take a snapshot of Mark's home directory

```
# zfs snapshot tank/home/marks@tuesday
```

- Roll back to a previous snapshot

```
# zfs rollback tank/home/perrin@monday
```

- Take a look at Wednesday's version of `foo.c`

```
$ cat ~maybe/.zfs/snapshot/wednesday/foo.c
```



ZFS Clones

- Writable copy of a snapshot
 - > Instantaneous creation, unlimited number
 - > Ideal for storing many private copies of mostly-shared data
 - Software installations
 - Workspaces
 - Diskless clients
- **Create a clone of your OpenSolaris source code**

```
# zfs clone tank/solaris@monday tank/ws/lori/fix
```



ZFS Send / Receive (Backup / Restore)

- Powered by snapshots
 - > Full backup: any snapshot
 - > Incremental backup: any snapshot delta
 - Very fast – cost proportional to data changed
- So efficient it can be used for remote replication

- Generate a full backup

```
# zfs send tank/fs@A >/backup/A
```

- Generate an incremental backup

```
# zfs send -i tank/fs@A tank/fs@B >/backup/B-A
```

- Remote replication: send incremental once per minute

```
# zfs send -i tank/fs@11:31 tank/fs@11:32 |  
ssh host zfs receive -d /tank/fs
```



Conclusion – ZFS is

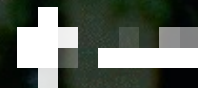
- Simple
 - > Concisely expresses the user's intent
- Powerful
 - > Pooled storage, snapshots, clones, compression, RAID-Z, universal storage, ...
- Safe
 - > Detects and corrects silent data corruption
- Fast
 - > Dynamic striping, intelligent prefetch, pipelined I/O
- Open
 - > <http://www.opensolaris.org/os/community/zfs>
- Free



Questions?



roman.strobl@sun.com



THANK YOU

Roman Strobl
OpenSolaris Evangelist

<http://blogs.sun.com/observatory>





Backup slides



Trends in Storage Integrity

- Uncorrectable bit error rates have stayed roughly constant
 - > 1 in ~12TB for desktop-class drives
 - > 1 in ~120TB for enterprise-class drives
- Bad sector every 8-20TB in practice
 - > Drive capacities doubling every 12-18 months
 - > Number of drives per deployment increasing
 - > → Rapid increase in error rates
- Silent and “noisy” data corruption more common
- Cheap flash storage even less reliable



Measurement in CERN

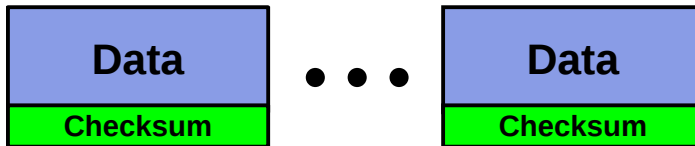
- Wrote a simple application to write/verify 1GB file
 - > Write 1MB, sleep 1 second, etc. until 1GB has been written
 - > Read 1MB, verify, sleep 1 second, etc.
- Ran on 3000 rack servers with HW RAID card
- After 3 weeks, found 152 instances of silent data corruption
 - > Previously thought “everything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



Reliability: End-to-End Data Integrity

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

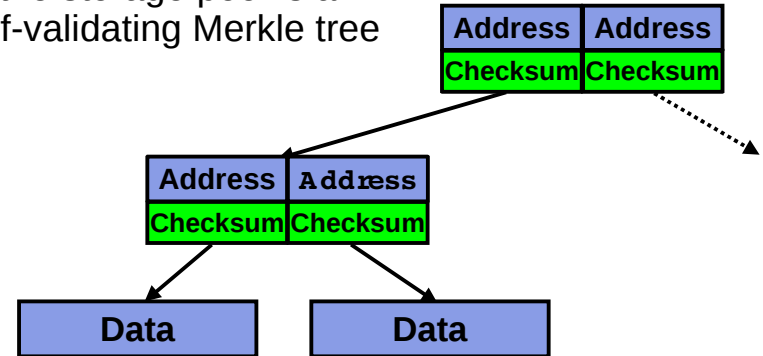


Disk checksum only validates media

✓	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



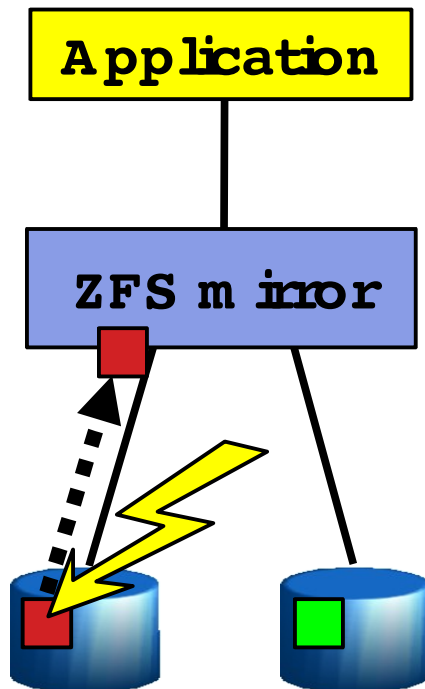
ZFS validates the entire IO path

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

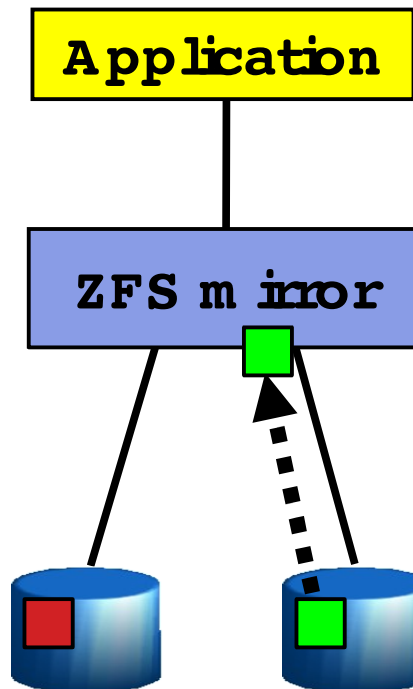


Reliability: Self-Healing Data in ZFS

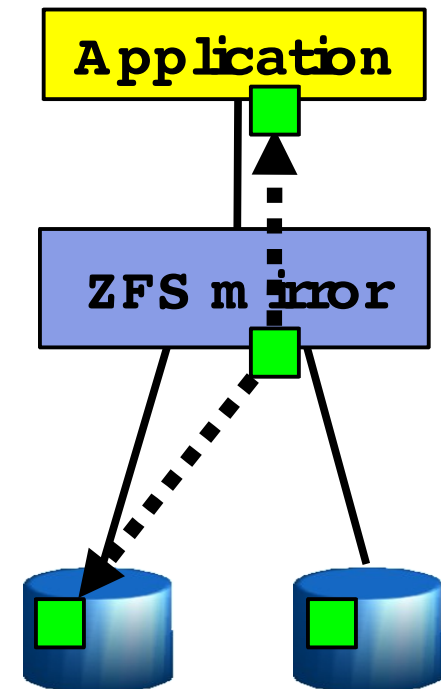
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



2. ZFS tries the second disk. Checksum indicates that the block is good.



3. ZFS returns good data to the application and repairs the damaged block.





Availability: RAID-Z

- Dynamic stripe width
 - > Variable block size: 512 – 128K
 - > Each logical block is its own stripe
- Both single- and double-parity
 - > raidz = raidz1, raidz2
- All writes are full-stripe writes
 - > Eliminate read-modify-write (it's fast)
 - > Eliminate the RAID5 write hole (no NVRAM)
- Detects and corrects silent data corruption
 - > Checksum-driven combinatorial reconstruction
- No special hardware – ZFS loves cheap disks



Availability: Easier Updates

- Less downtime with ZFS when doing updates!
 - > Make a snapshot
 - > Install a package
 - > If the update fails you can rollback instantaneously
- Integration of ZFS and IPS
 - > Update OpenSolaris using “pkg image-update”
 - > Snapshots are created for all filesystems
 - > New boot environment is created
 - > If upgrade fails you can just boot into old environment

ZFS Performance

- Copy-on-write design
 - > Turns random writes into sequential writes
- Multiple block sizes
 - > Automatically chosen to match workload
- Pipelined I/O
 - > Maximum possible I/O parallelism
- Dynamic striping across all devices
- Intelligent prefetch



ZFS Scalability

- 7 • Immense capacity (128-bit)
 - > ZFS capacity: 256 quadrillion ZB (1ZB = 1 billion TB)
 - > Exceeds limits of Earth-based storage
- 100% dynamic metadata
 - > No limits on files, directory entries, etc.
 - > No wacky knobs (e.g. inodes/cg)
- 4 • Concurrent everything
 - > Byte-range locking: parallel read/write without violating POSIX
 - > Parallel, constant-time directory operations

ZFS Administration

- Hierarchical filesystems with inherited properties
 - > Filesystems become administrative control points
 - > Control compression, checksums, quotas, reservations, and more
 - > Delegate administrative privileges to ordinary users



ZFS Data Migration

- Host-neutral on-disk format
 - > Change server from x86 to SPARC, it just works
 - > Adaptive endianness: neither platform pays a tax
 - Writes always use native endianness, set bit in block pointer
 - Reads byteswap only if host endianness != block endianness
- ZFS takes care of everything
 - > Forget about device paths, config files, /etc/vfstab, etc.
 - > ZFS will share/unshare, mount/unmount, etc. as necessary
- Export pool from the old server

```
old# zpool export tank
```

- Physically move disks and import pool to the new server

```
new# zpool import tank
```



ZFS Delegated Administration

- Ability to delegate zfs(1M) administrative tasks to normal users
- 'zfs allow'
- 'zfs unallow'
- Grant privileges to a specific user

```
# zfs allow marks create,snapshot tank/marks
```

```
# zfs unallow marks create,snapshot tank/marks
```