ZFS In Business

Roch Bourbonnais
Sun Microsystems
Roch.Bourbonnais@sun.com
What is ZFS

- Integrated Volume and Filesystem w no predefined limits
- Volume Management
  - pooling of disks, luns... in raid-z and mirrored protection
- Pooled Blocks in a Tree Structure
  - Transactional
  - Checksum and self healing
  - Copy on write
  - Flash aware
  - Constant time snapshot
- Feature rich Filesystem
  - Compression, encryption, quota/reservation, acl, mixed case, etc
- Contextual I/O Scheduling
Many Storage Models using ZFS

• ZFS as Filesystem on Host
  > ZFS on JBODS
  > ZFS on JBODS + flash/NVRAM acceleration
  > ZFS on Storage Array
  > ZFS on 100% flash

• NFS/CIFS as Filesystem on Linux, Solaris, Proprietary OS
  > backed by ZFS in IP network

• iSCSI on Linux, Solaris, Proprietary OS
  > backed by ZFS in IP network
  > raw dev or filesystem on top
Copy-on-Write and Transactional

Initial block tree

Original Data

New Data

Uber-block

Copy-on-write of indirect blocks

Original Pointers

New Pointers

Rewrites the Uber-block

New Uber-block
End-to-End Checksums

Checksums are separated from the data

Entire I/O path is self-validating (uber-block)

Prevents:
> Silent data corruption
> Panics from corrupted metadata
> Phantom writes
> Misdirected reads and writes
> DMA parity errors
> Errors from driver bugs
> Accidental overwrites
Design Center of ZFS

• ZFS is transactional and never overwrites live data
  > no risk of partial writes
  > writes are always streaming to storage
  > instantaneous snapshot

• Storage Pool architecture
  > trivial provisioning of blocks (quotas and reservations)
    – grow and shrink ZFS at will, instantaneously
  > manage large number of snapshots
    – user does own recovery
  > send/recv only diffs to keep two projects replicated

• ZFS is free: Open and Gratis
  > MacOS, FreeBSD, Nexenta,..
Zpool, ZFS and Zvols

• Disks or Luns are pooled in a protection level
  > stripped, mirrored, raid-z : Zpool

• **Zpool must be attached to a single host**

• Zpool can grow number of devices
  > add mirrored pairs or full raid-z groups
  > Removing a device requires replacing it with (virtually) bigger one

• ZFS and Zvols
  > your lightweight administrative point (mountpoint, snapshots, quota, reservation)
  > ZFS stores files, Zvol stores blocks
  > NFS shares, iSCSI target
ZFS never overwrites live data nor metadata

- Overwritting Metadata means we'd need to fsck
- Overwritting live data: risks of application corruption
- Not overwritting data|metadata means writes can go anywhere
  > coupled with I/O scheduler means writes go at write streaming speed, even random writes
  > writes aggregates small I/O into bigger ones
  > can store more data with less disk occupancy freeing them up to service reads
  > reads can be prefetched
    - latency is preserved
ZFS instantaneous snapshots

- Data is organised as a tree of blocks and pointers
  > each block has birth time and death times
- Taking a snapshot of a FS just means not deleting the top pointer and storing the list of deleted block pointers
  > constant time; part of the transaction group pool update
- Deleting a snapshot involves walking the list of deleted blocks in this snapshot and either
  > free them if previous snapshot time before block birth
  > otherwise tag to the previous snapshot's deadlist
  > Cost proportional to blocks deleted in this snapshot
- Similar technique to identify all blocks modified between 2 arbitrary snapshots (zfs send)
  > Flexible backup architecture, Storage replication, User recovery
ZFS instantaneous snapshots

http://blogs.sun.com/erwann/entry/zfs_on_the_desktop_zfs
ZFS instantaneous snapshots

http://blogs.sun.com/erwann/entry/zfs_on_the_desktop_zfs
ZFS Clones

• Just duplicate a FS master node from snapshot
  > block deletes only looks at blocks born in the clone
  > Master and any of it clones can be interchanged
  > The Master, it's Snapshot and Clones are interdependant

• Create 100s of new Virtual Desktop Instances with ~0 space
• Create mysql slaves
• http://blogs.sun.com/ahrens/entry/is_it_magic
ZFS Records: small files

- Small files (<128K) stored as single record (coherency unit)
- Single I/O to read or update file
- **No need to tune recordsize for smallfiles**
- Writes to small files allocate to contiguous blocks and aggregate in the I/O scheduler

  1k Files created and Ios Per Minute
  
  - Sample Create IOs
  - #1 214643 2856
  - #2 215409 3342
  - #3 212797 2917
  - #4 211545 2999

ZFS Records : big files

• Big files are stored as 128K records
• **No need to tune recordsize for big files**
• Can be more CPU efficient
• Faster to build up caches
ZFS Records : DB

- DB files are access randomly at fixed records, 4K, 8K, 16K depending on DB
  > need to tune recordsize for DB data files
  > but not for log and temp files (accessed sequentially)
- 1 I/O to get any DB records
- writes will aggregate in ZIL (O_SYNC, FSYNC)
- background writes will aggregate in main storage pool
- ZFS write strategy leads to less I/O occupancy freeing up the spindles to serve reads
Synchronous Write Performance

- Handled through the ZFS Intent Log: ZIL.
- The ZIL does not have a major issue.
- Single threaded NFS or O_SYNC workloads are slow on any FS.
- Single threaded NFS or O_SYNC workloads benefit from low latency storage.
- The ZIL can now (S10U6) target separate devices:
  - Isolate ZIL commits from reads and background writes.
  - Great for NVRAM or SSD based luns.!!!
  - Great NFS accelerator.
ZFS and CPU

• checksums helps data integrity but also metadata
  > can be turned off for data
  > considered cheap on CPU

• Pool transaction is currently CPU intensive
  > occurs every 5-30 seconds depending on load
    – should lead to 1-5 seconds of I/O each TXG

• monitor CPU usage and provision for ZFS's share

• CPU in 7000 or NFS/CIFS server is not competing with Application
  > NFS directio with ethernet Jumbo frame helps reduce CPU in NFS
ZFS Data Protection Models

- ZFS Mirroring
  - writes are done twice in case one copy is corrupted on way to storage
  - reads go to one half
  - automatic self-healing of data blocks!

- ZFS Raid-Z
  - Delivers good performance for streaming, but random reads need spindles.
  - works on JBODS and is safe unlike software raid-5
  - self-healing.

- Storage based protection
  - reuses existing infrastructure
  - ZFS writes architecture means everything is full stripes in backend :-)
  - Self-healing avail for ZFS metadata but not application data
Many Storage Models using ZFS

• ZFS as Filesystem on Host
  > $ave on FS software
  > buy CPU, memory or spindles instead

• ZFS on JBODS
  > raid-z for pure streaming workload or pure capacity ($/GB)
  > mirror otherwise

• ZFS on JBODS + flash/NVRAM acceleration
  > flash/NVRAM to host ZIL, synchronous writes
  > Can run DB but more variability than traditional.
  > Great for NFS, CIFS, iSCSI

• ZFS on Storage Array
  > expensive but fast, no data self-healing, failmode=wait
Many Storage Models using ZFS

- **ZFS on 100% flash**: need to mirror, ZFS checksum a plus.
- **NFS/CIFS backed by ZFS in IP network**
  - reduce CPU consumption with NFS directio and Jumbo Frames
  - A lot of ZFS benefits
  - common tasks speeded up by flash/nvram based ZIL
- **iSCSI as NAS Storage on Host, using FS or Not**
  - backed by ZFS in IP network with Jumbo
  - Requires some flash/nvram based ZIL.
  - Exchange/iSCSI, DB on raw, VDI
- **Convergence of blocks and files using ZFS**
7000 Storage Line

Figure 1. Sun Storage 7000 Unified Storage System family.
Sun Storage 7000 Series

• A new line of network storage products developed by Sun's Fishworks team, ranging from 2TB to 288TB, single-node to active/active cluster

• Addresses key pain-points in enterprise storage:
  > Too expensive
  > Too slow
  > Too opaque
  > Too difficult to manage

• As such, represents a quantum leap in price/performance and a disruptive innovation in enterprise storage
The Analytics Advantage

- Real-time system performance visualization supporting *ad hoc*, high-level queries:
Smugmug & 7000 Storage Line

- 450 M full resolution photo sharing
- LAMP, run by 2 Ops, Amazon, Akamai
- MySQL, Memcached, 4core systems, 64GB to 2TB of storage
- ext3 down, ZFS up
- ZFS compression 1.5X faster!
- Solaris not enough like linux
- OpenStorage
  - ZFS on any client with SSD
  - NFS, iSCSI, CIFS, HTTP, ftp,
  - Massive flexibility
  - Analytics a game changer
Eating tasty Toro

✧ Crazy fast. 9.6K iops, 4.5K under 43us, 8K under 166us

Protocol: NFSv3 operations per second of type write broken down by latency

Range average:

- 9624 ops per second

2008-12-16
21:16:10
21:16:15
21:16:20
Smugmug & 7000 Storage Line

- Everything Runs better
  - revision control with snapshots
  - stateless linux clients
  - email
  - Developper Home
  - 7000 replication features

- http://blogs.smugmug.com/don
Hybrid Storage Pool: Optimized with Flash

  > Adam Leventhal

- ZFS handles critical Commits through the ZIL
  > ZIL on SSD with dual attachment
    - cluster failover gets to read the log directly in the storage
    - no data flow between cluster heads
  > ZIL data is never read except after outage
    - flash firmware write optimized in 7000 series
    - each device deliver 100MB/s+ of synchronous writes
    - data automatically freed on ZFS transaction (~30s)
Hybrid Storage Pool: Optimized with Flash

- Using flash for primary data store can be expensive
  - requires mirroring
- Flash as a cache is cheap.
  - ZFS can hold on to data evicted from primary cache: L2ARC
  - flash firmware read optimized
  - deliver low latency reads to your critical datasets
  - Capacity and Read IOPS more important than write speed
Hybrid Storage Pool : Optimized with Flash

Sun Storage 7410

Read IOPS increased by 500% with the L2ARC

http://blogs.sun.com/brendan/entry/l2arcScreenshots
How to get terrible performance

• Run against storage array that flush caches
• Run simple benchmarks without decoding the numbers
  > compare write to cache vs write to disk
• Run the pool at 95% disk full
• Do random reads from widest raid-z
• Run a very large DB without tuning the recordsize
• Don't provision enough CPU
• Don't configure swap space
• Don't read the ZFS Best Practices
How to get Great performance

- small files (<128K)
  - ufs allocates 1 inode per MB
  - netapps 1 / 32K
  - ZFS uses 1.2K to store 1K files !!!
  - Create 10s of files per single I/Os
  - $ miss reads == single disk I/O

- ZFS does constant time snapshot
  - it's basically a noop to take a snapshot
  - snap deletions proportional to changes
  - snapshots helps simplify your business
How to get Great performance

• Run ZFS in the storage back end (7000 Storage)
• Or provision for CPU usage.
• Configure enough RPM
  > 2 Mirrored 7.2 K RPM vs 1 x 15 K RPM in Raid-5
• Move Spindle Constrained setup to ZFS
  > write streaming + I/O aggregation
    > efficient use of spindles on writes,
    > 100% full stripes in storage
  > free spindles for reads
  > use a separate intent log (NVRAM or SSD or just N separate spindles) for an extra boost
Solaris Update 6

- Finally got write throttling, ZFS won't eat all of memory
  > Grows and shrink dance now as designed
  > Capping the ARC seems commonly done
  > ZFS reports accurate freemem, others cache data in freemem

- Cache flushes to SAN array partially solved
  > HDS, EMC with recent firmware are ok.
  > Can be tuned per array
  > Others? set zfs_nocacheflush (cf evil tuning guide)

- Vdev level prefetching is auto tuning
  > no problems there
Solaris Update 6

• We have the separate intent log
  > one or a few disks, but preferably SSD or NVRAM/DRAM device

Upcoming

• L2 ARC
  > on/off per dataset

• ARC
  > on/off per dataset, ~directio

• Storage 7000
  > Tracks Nevada
Tuning is Evil

• Leave a trace, explain motivation
  > zfs_nocacheflush (on storage arrays that do)
  > capping the ARC (to preserve large pages)
  > zfs_prefetch_disable (zfetch consuming cpus)
  > zfs_vdev_max_pending (default 35, 10-16 for DB)
  > zil_disable (NO!!! don't or face application corruptions)

• No tuning required
  > vdev prefetch (issue now fixed)
ZFS Best Practices

- Tune recordsize only on fixed records DB files
- Mirror for performance
- 64-bit kernel (allows greater ZFS caches)
- configure swap (don't be scared by low memory)
- Don't slice up devices (confuses I/O scheduler)
- For raid-z[2] : don't go two wide (for random reads)
- Isolate DB log writer if that is critical (use few devices)
- Separate Root pool (system's identify) and data pools (system's function)
ZFS Best Practices

• Don't mix legacy and non legacy shares (it's confusing)
• 1 FS per user (1 quota/reserv; user quota are coming)
• Rolling Snapshots (smf service)
• Instruct backup tool to skip .zfs
• Keep pool below 80% full (helps COW)
MySQL Best Practices

- Match Recordsize with DB (16K)
- Use a separate intent log device within main zpool
- Find creative use of Snapshot/Clones send/recv
  - backups
  - master & slave architecture
- Use the ARC and L2ARC instead of disk RPM
  - a caching 7000 series serving masters & slaves
- NFS Directio and Jumbo Frames
  - save CPU cycles and memory for application
    - Set innodb_doublewrite=0
- Linux
  - innodb_flush_method = O_DIRECT
  - echo noop > /sys/block/sde/queue/scheduler
Tuning & Best Practices

• Tuning and BP wikis

• ZFS Dynamics : In-Depth view
  > http://blogs.sun.com/roch/entry/the_dynamics_of_zfs

• Blue Prints
  > http://wikis.sun.com/display/BluePrints/Main

• Performance Savvy Bloggers
  > joyent (Ben Rockwood), Smugmug (Don Mcaskill), Neel (Oracle and MySQL), Media Temple

• MySQL People
  > Allan Packard, David Lutz, Neel Nagdir